

J. Hegner

GRAFIK IN MASCHINENSPRACHE AUF DEM COMMODORE 64

Grafik-Programme –
Vergleich und Zusammenarbeit
von BASIC und Maschinensprache

Programme auf Diskette/Kassette erhältlich



J. Hegner

GRAFIK IN MASCHINENSPRACHE AUF DEM COMMODORE 64

Grafik-Programme –
Vergleich und Zusammenarbeit
von BASIC und Maschinensprache



CIP-Kurztitelaufnahme der Deutschen Bibliothek

Hegner, Jürgen:
Grafik in Maschinensprache auf dem Commodore 64:
Grafik-Programme, Vergleich und Zusammenarbeit
von BASIC und MASCHINENSPRACHE/J. Hegner. –
Vaterstetten: IWT, 1984
ISBN 3-88322-051-5

ISBN 3-88322-051-5
1. Auflage 1984

Printed in Western Germany

Druck: FGB
Umschlaggestaltung: Kaselow und Partner, München

Vorwort:

Der Commodore bietet für seine Preisklasse unter 800 DM eine Menge an Grafikmöglichkeiten. Was bis vor kurzem nur sehr teure Computer konnten, ist jetzt für jedermann erschwinglich. Nun ist aber die Grafikprogrammierung ein recht umfangreiches Kapitel, das vielleicht etwas undurchschaubar anmutet. Dies ist aber keineswegs der Fall. Bisher hat nur die richtige Information gefehlt. Dieses Buch baut auf geringe Grafikkentnisse auf, soll weiterführen und Ausblicke vermitteln. Besonders ist die Maschinensprache interessant, da BASIC und komplizierte Berechnungsverfahren oft am Geduldsfaden zehren. Von einfachen Maschinenprogrammen, die leicht durchschaubar sind und nach den vorangegangenen BASIC Programmen erstellt wurden, geht es bis zu anspruchsvollen Grafikhilfsprogrammen hin, die die zeitraubende Erstellung von Grafiken auf ein Minimum verkürzen. Alle programmierbaren Betriebsarten des Grafikbausteins im Commodore 64 werden ausführlich besprochen und mit Abbildungen oder Programmen belegt.
Ferner werden schnelle Maschinenprogramme zum Punkt- oder Linienzeichnen im hochauflösenden Grafikbildschirm aufgezeigt. Somit können komfortable Eingabemöglichkeiten programmiert werden.
Im Anhang wird der IWT SPRITE KOMFORT KIT besprochen und seine ca. 40 neuen Befehle in einer Liste dargestellt. Weiterhin findet man im Anhang eine Tabelle mit allen zur Verfügung stehenden Registern und den Funktionen des Video Interface Controllers, der im Computer für alle Grafikangelegenheiten zuständig ist.

Der Autor — Anzing, April 1984

Inhaltsverzeichnis:

1. Einleitung

Der Commodore 64 ist in Sachen Grafik so etwas wie ein kleines Wunderwerk. Die Verantwortung dafür trägt der sogenannte VIC Chip 6567 (VIC=Video Interface Chip). Bei richtiger Programmierung des Bausteins ermöglicht er alles was einem gerade so einfällt. Angefangen vom Standardformat, das aus 25 Zeilen mit je 40 Zeichen besteht, über die hochauflösende Grafik, mit der man über 320 x 200 einzeln ansprechbare Punkte verfügt, bis hin zu den sogenannten Sprites. Das Wort Sprite würde wörtlich ins Deutsche übersetzt soviel wie Geist oder Kobold heißen. Diese Bezeichnung ist gar nicht so unzutreffend, denn Sprites sind kleine einfach zu bewegende und freiprogrammierbare Figuren. Pro Sprite sind 24 x 21 Einzelpunkte vorgesehen. Mit Hilfe dieser bewegbaren Gestalten sind zum Beispiel Spielprogramme, die auf den älteren Commodore Computern schier unmöglich zu programmieren schienen, relativ einfach zu erstellen. Weiterhin verfügt der Commodore 64 über eine Menge verschiedener Farben. Die hochauflösenden Grafiken, der Zeichensatz und die Sprites können selbstverständlich auch mehrfarbig gestaltet werden. Der Video Chip kann auch so programmiert werden, daß die obere Hälfte aus hochauflösender Grafik und die untere Hälfte aus dem normalen Textmodus besteht.
Die einzelnen zu erreichenden Zustände werden in den nachfolgenden Kapiteln ausführlich in BASIC und Maschinensprache erläutert. Des weiteren werden einige Grafik Hilfsprogramme gezeigt, die das Arbeiten mit der hochauflösenden Grafik wesentlich erleichtern.
Um Ihnen das lästige und fehlerbehaftete Eintippen der oft längeren Beispielprogramme zu ersparen, können Sie diese Programme auf Kassette oder Diskette beziehen.

2. Grafik

Wenn Sie Ihren Commodore 64 einschalten, dann wird der Video Interface Controller (VIC) mit Standardwerten versehen. Diese Standardwerte sind zum Beispiel der Wert 14 für die Rahmenfarbe und der Wert 6 für die Hintergrundfarbe. Die Standardwerte werden ebenfalls eingespeichert, wenn Sie die Tastenkombination RUN/STOP-RESTORE benutzen.
Dieser RUN/STOP-RESTORE Modus ist dann hilfreich, wenn Sie beim Experimentieren mit der Grafik des öfteren 'komische' Zeichen auf Ihrem Bildschirm sehen. Entweder benutzen Sie dann RUN/STOP-RESTORE oder schalten den Computer kurz aus. Danach ist der Einschaltzustand in Bezug auf die Grafik wiederhergestellt.
In den folgenden Kapiteln werden oftmals die einzelnen Register des Video Chips benutzt. Ist eine solche Adressenangabe vorhanden, wird sie zuerst dezimal (z.B. 53248) und dann hexadezimal (z.B. $D000) angegeben. Zur besseren Unterscheidung ist man übereingekommen vor hexadezimalen Zahlen das Dollar- oder Stringzeichen ($) zu setzen.
Die Beispielprogramme in Maschinensprache sind, des besseren Verständnisses wegen, nach den Basicprogrammbeispielen erstellt. So hat man den direkten Vergleich zwichen den beiden Programmierarten. In den hinteren Kapiteln werden dann Programme zum Punktzeichnen erläutert. Hier spielt der Faktor Geschwindigkeit eine schwerwiegende Rolle. In jedem Fall ist das Maschinenprogramm das schnellere.
Die Maschinenprogramme können Sie entweder über einen sogenannten Monitor direkt in Maschinensprache oder über ein BASIC- Einleseprogramm mittels DATA Zeilen eingeben.

2.1 Speicherbereiche

Unmittelbar nach dem Einschalten werden die Speicherbereiche mit ihren Standardwerten versehen. Das bedeutet, daß das VIDEO-RAM (RAM in dem die Werte der Zeichen stehen die auf dem Bildschirm erscheinen) bei Adresse 1024 ($0400) beginnt und bei 2023 ($07E7) endet. Das Video-RAM besteht aus 1000 Bytes, denn der Bildschirm setzt sich im Standardformat von 25 Zeilen mit je 40 Zeichen ebenfalls aus 25 x 40 = 1000 Zeichen zusammen. Zu jedem Zeichen im Video-RAM gibt es eine bestimmte Farbe. Die Farbwerte stehen im FARB-RAM. Das Farb-RAM besteht auch aus 1000 Zeichen und reicht von 55296 ($D800) bis 56295 ($DBE7). Das erste Byte des Video-RAM's 1024 ($0400) bezieht sich auf das erste Byte im Farb-RAM 55296 ($D800). Der Commodore 64 besitzt insgesamt 16 Farben, d.h. im Farb-RAM sind nur die untersten 4 Bit ausschlaggebend (XXXX 0001 z.B. Wert 1 für die Farbe weiß). Die Farben können also Werte von 0 ($00) bis 15 ($0F) annehmen. Ganz anders sieht die Sache im Video-RAM aus. Im Video-RAM werden alle 8 Bit benötigt, denn es gibt ja 256 Zeichen, also Werte von 0 ($00) bis 255 ($FF).
Die Programmierung der verschiedenen Grafikzustände des Video Chips werden durch 47 Kontrollregister gesteuert. Die Kontrollregister beginnen bei Adresse 53248 ($D000) und reichen bis 53294 ($D02E). In den folgenden Kapiteln werden alle Register eingehend beschrieben. Weiterhin gibt es im Anhang eine Tabelle, in der alle Register übersichtlich zusammengestellt sind und ihre Funktion kurz beschrieben wird.

2.2 Auswahl der 16k Speicherbereiche

Eine Auswahl des 16k Speicherbereiches muß deswegen getroffen werden, weil der Video Chip nur

16k auf einmal ansprechen kann. Im Standardzustand sind die niedrigsten 16k angesprochen, also der Bereich von 0-16383 ($0000-$3FFF). Durch Änderung des 16k Blocks ist es möglich, jeden anderen der insgesamt vier Blöcke anzusprechen. Die Auswahl erfolgt aber nicht über ein Kontrollregister des Video Chips, sondern über zwei Bits im 6526 Complex Interface Adapter Chip 2 (CIA 2). Bit 0 und Bit 1 der Adresse 56576 ($DD00) kontrollieren die Auswahl der Bereiche. Bevor man aber diese beiden Bits ändert, muß man sich vergewissern, daß die Datenrichtung dieser Bits auf Ausgang bzw. 1 steht. Das Datenrichtungsregister befindet sich in 56578 ($DD02). Für unseren Zweck müssen Bit 0 und Bit 1 jeweils auf 1 sein. Das folgende Beispiel erläutert den obigen Zusammenhang:

BASIC:

```
POKE 56578,PEEK (56578) OR 3
; setzt Datenrichtung auf Ausgang
POKE 56576,(PEEK (56576) AND 252) OR Wdez
; wählt  16k Block an, Wdez aus Tabelle
```

MASCHINENSPRACHE:

```
    DATRICH =$DD02
    BLOCK   =$DD00
LDA DATRICH ; Datenrichtung laden
ORA #$03    ; Bit 0 und Bit 1 setzen
STA DATRICH ; Datenrichtung auf Ausgang
LDA BLOCK   ; Blockwert laden
AND  #$FC   ; Bit 0 und Bit 1 löschen
ORA #$Whex  ; Whex aus Tabelle
STA $BLOCK  ; Blockanwahl
RTS
```

Wdez bzw. Whex kann folgende Werte annehmen:

Wdez	0	1	2	3
Whex	$00	$01	$02	$03
Bit0,1 56576 ($DD00)	00	01	10	11
Bereich	3	2	1	0
Start-adresse	49152 $C000	32768 $8000	16384 $4000	0 $0000

Der Bereich 0, d.h. Wert W = 3 ($03), ist der Standardzustand. Diese Auswahl ist von großer Bedeutung. Man muß sich beim Arbeiten mit der Grafik immer vergewissern, welcher der vier 16k Blöcke gerade angesprochen ist. Der Video Chip nimmt seine komplette Information aus diesem Bereich. Hierunter fällt zum Beispiel das Bitmuster für die Sprites oder der Bereich für die hochauflösende Grafik. Später werden wir sehen, daß man in Verbindung mit der hochauflösenden Grafik am besten den Bereich 0 anwählt. Die Gründe dafür werden in den späteren Kapiteln erläutert.

2.3 Video-RAM

Das Video-RAM ist der Speicher in dem die Zeichen stehen, die an einer bestimmten Stelle auf dem Bildschirm erscheinen. Der Bereich des Video-RAM's kann mit der Adresse 53272 ($D018) verschoben werden. Für die Position des Video-RAM's werden aber nur die oberen 4 Bit benötigt. Bit 4...7 bestimmen die Lage des Video-RAM's.

Um die Position des Video-RAM's zu verschieben dienen die folgenden Programme:

BASIC:

```
POKE 53272,(PEEK (53272) AND 15) OR Wdez
; verschiebt die Lage des Video-RAM's
```

MASCHINENSPRACHE:

```
    POSVID =$D018
LDA POSVID  ; Position des Video-RAM's laden
AND #$0F    ; Bit 4...7 löschen
ORA #$Whex  ; Bit 4...7 entsprechend setzen
STA POSVID  ; Position des Video-Ram's verschieben
RTS
```

Wdez bzw. Whex kann folgende Werte annehmen:

Wdez	Whex	Bits 4...7 53272 ($D018)	Video-Ram Position dezimal	hexadezimal
0	$00	0000 XXXX	0	$0000
16	$10	0001 XXXX	1024	$0400
32	$20	0010 XXXX	2048	$0800
48	$30	0011 XXXX	3072	$0C00
64	$40	0100 XXXX	4096	$1000
80	$50	0101 XXXX	5120	$1400
96	$60	0110 XXXX	6144	$1800
112	$70	0111 XXXX	7168	$1C00
128	$80	1000 XXXX	8192	$2000
144	$90	1001 XXXX	9216	$2400
160	$A0	1010 XXXX	10240	$2800
176	$B0	1011 XXXX	11264	$2C00
192	$C0	1100 XXXX	12288	$3000
208	$D0	1101 XXXX	13312	$3400
224	$E0	1110 XXXX	14336	$3800
240	$F0	1111 XXXX	15360	$3C00

Der Standardwert für die Position des Video-RAM's ist eingestellt, wenn der Wert W = 16 ($10) beträgt. Ist dies der Fall, so beginnt das Video-RAM bei Adresse 1024 ($0400).
Wenn Sie in einem anderen 16k Bereich als dem Bereich 0 (vgl. 2.2) arbeiten, müssen Sie die Startadresse des 16k Bereiches zu der Position des Video-RAM's addieren.
Wollen Sie mit verschobenem Video-RAM so wie im Normalzustand arbeiten, müssen Sie das dem Betriebssytem mitteilen. Das Register 648 ($0288) kontrolliert das. In ihm muß die Page (= Seite, eine Page beinhaltet 256 Byte) stehen, in der das Videoram liegt. Wenn also zum Beispiel das Video-RAM bei 3072 ($0C00) beginnt, so benötigt das Betriebssystem die Information, daß sich das Video-RAM in der Page 3072 : 256 = 12 befindet. Die Weitergabe der Werte an das Betriebssystem erfolgt so:

BASIC:

```
Wdez = 3072 : 256 = 12        ; errechnen der Page
POKE  648,Wdez                ; Wert übergeben
```

MASCHINENSPRACHE:

```
Whex = 3072 : 256 = 12 ($0C) ; errechnen der Page

    VIDCTRL =$0288
LDA #$Whex
STA VIDCTRL ; Wert übergeben
RTS
```

2.4 Farb-RAM

Das Farb-RAM kann im Gegensatz zum Video-RAM nicht verschoben werden. Die feste Position reicht von Adresse 55296 ($D800) bis 56295 ($DBE7). Im Farb-RAM steht die Farbe eines Zeichens auf dem Bildschirm bzw. im Video-RAM. Ist der Einschaltzustand in Bezug auf die Speicherbereiche hergestellt, so befindet sich das Video-RAM ab Adresse 1024 ($0400). Das erste Byte im Video-RAM bezieht sich auf das erste Byte im Farb-RAM, d.h. Video-RAM-Adresse 1024 ($0400) hat die Farbe, die in der Farb-RAM-Adresse 55296 ($D800) steht. Adresse 1025 ($0401) bezieht sich auf 55297 ($D801) usw. Da der Commodore 64 über 16 Farben verfügt, werden vom Farb-RAM nur die untersten 4 Bit benötigt. Ein Byte im Farb-RAM sieht so aus: XXXX 0010 ist das Bitmuster für rot. Der Farbwert für rot ist 2.
Lassen Sie uns nun ein Beispiel betrachten. Es soll in die linke obere Bildschirmecke ein weißes A geschrieben werden und zwar nicht mittels PRINT-, sondern mittels POKE-Befehlen:

BASIC:

```
POKE 1024,1    ; Zeichen A setzen
POKE 55296,1   ; Farbe des Zeichens ist weiß
```

MASCHINENSPRACHE:

```
    ZEICHEN =$0400
    FARBE   =$D800
LDA #$01     ; Zeichenwert 1 laden (=A)
STA ZEICHEN  ; Zeichen im Video-RAM abspeichern
LDA #$01     ; Farbe ist weiß  (Farbwert 1)
STA FARBE    ; Farbe im Farb-RAM abspeichern
RTS
```

2.5 Zeichengenerator

Der Zeichengenerator auch Charaktergenerator genannt, ist beim Commodore 64 ein Kapitel für sich. Es ist ganz offensichtlich nicht so einfach ihn zu verändern und damit eigene Zeichen zu definieren. Dennoch ist es möglich, wie wir später noch sehen werden. Dann können wir unsere eigenen- zum Beispiel griechische-Zeichen definieren. Auf der Programmkassette bzw. Diskette zu diesem Buch befindet sich ein Programm, mit dem Sie mittels Joystick komfortabel Ihren eigenen Zeichensatz gestalten können.
Im Normalzustand, d.h. wenn mit dem Standardzeichensatz gearbeitet wird, erhält der Video Chip die Information -zum Beispiel wie ein 'A' aussieht- aus dem Zeichengenerator. Der Zeichengenerator kann ähnlich dem Videoram verschoben und damit verändert bzw. neu definiert werden. Der Charaktergenerator enthält für einen Zeichensatz 2048 Bytes (2k). Da beim Standardzeichensatz 256 verschiedene Zeichen existieren, stehen demnach im Zeichengenerator 256 x 8 = 2048 Bytes zur Verfügung. Das bedeutet, daß man für ein Zeichen 8 Byte mit je 8 Bit benötigt. Unsere Zeichen bestehen ja aus 8 Zeilen mit 8 Punkten. Den genaueren Aufbau des Zeichengenerators betrachten wir noch im Kapitel Standard Character Mode.
Jetzt wollen wir uns aber um die Position kümmern. Zuständig für die Position des Zeichengenerators ist Bit 1...3 der Adresse 53272 ($D018). Bit 0 hat keinen Einfluß auf die Position. Die oberen 4 Bit, Bit 4...7 also, haben wir bereits im Kapitel Video-RAM kennengelernt. Sie bestimmen nämlich die Lage des Video-RAM's. Das folgende Programm dient zur Bestimmung der Stelle an der sich der Zeichengenerator befindet. In der anschließenden Tabelle werden alle erreichbaren Positionen aufgezeigt.

BASIC:

```
POKE 53272,(PEEK (53272) AND 240) OR Wdez
; verschiebt den Zeichengenerator
```

MASCHINENSPRACHE:

```
    POSCHR =$D018
LDA POSCHR  ; Position laden
AND #$F0    ; Bit 0...3 löschen
ORA #$Whex  ; Bit 1...3 entsprechend setzen
STA POSCHR  ; Neue Lage abspeichern
RTS
```

Wdez bzw. Whex kann folgende Werte annehmen:

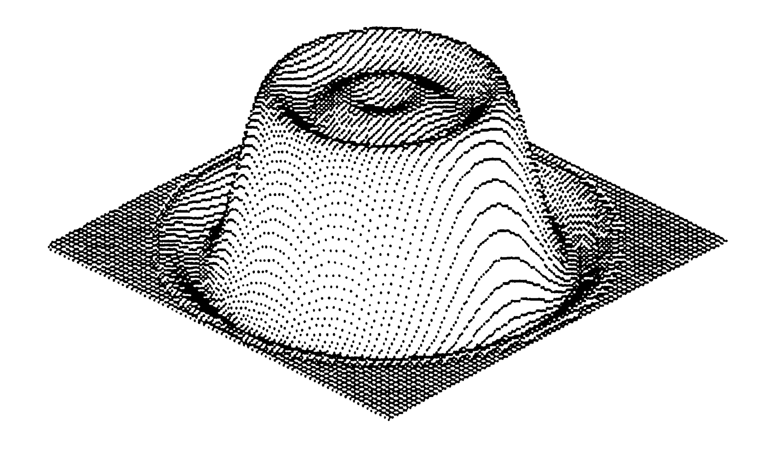
Wdez	Whex	Bits 1...3 53272($D018)	Zeichengenerator Position dezimal	hexadezimal
0	$00	XXXX 000X	0	$0000
2	$02	XXXX 001X	2048	$0800
4	$04	XXXX 010X	4096	$1000
6	$06	XXXX 011X	6144	$1800
8	$08	XXXX 100X	8192	$2000
10	$0A	XXXX 101X	10240	$2800
12	$0C	XXXX 110X	12288	$3000
14	$0E	XXXX 111X	14336	$3800

Der Normalzustand ist erreicht wenn der Wert W = 2 ($02) beträgt.
Der Zeichengenerator liegt in Wirklichkeit, d.h. im ROM, im Bereich 53248-57343 ($D000-$DFFF) vor. Der Commodore 64 arbeitet mit dem Zeichengenerator aber, nur wenn sein Bild (ROM IMAGE) im Bereich 4096-8191 ($1000-$1FFF) oder im Bereich 36864-40959 ($9000-$9FFF) liegt. Das hängt damit zusammen, daß der Zeichengenerator des Standardzeichensatzes nur dann angesprochen werden

kann, wenn als 16k Bereich der Bereich 0 oder 2 gewählt wird. Beim Commodore 64 gibt es genau zwei Standardzeichensätze. Die Aufgliederung der Bereiche, d.h. wie der Zeichensatz im einzelnen aufgebaut ist, geht aus folgender Tabelle hervor:

Satz	Startadresse dez.	hex.	Abbildung ROM IMAGE		Bemerkung
0	53248	$D000	4096	$1000	Großchrift
	53760	$D200	4608	$1200	Grafikzeichen
	54272	$D400	5120	$1400	Reverse Großchrift
	54784	$D600	5632	$1600	Rev. Grafikzeichen
1	55296	$D800	6144	$1800	Kleinschrift
	55808	$DA00	6656	$1A00	Grafikzeichen + Großchrift
	56320	$DC00	7168	$1C00	Reverse Kleinschrift
	56832	$DE00	7680	$1E00	Rev. Grafikzeichen + rev. Großchrift

Wie bereits gesagt, liegt der Zeichengenerator, im ROM, ab Adresse 53248 ($D000) an aufwärts. Seine Abbildung genannt ROM IMAGE liegt aber im Normalzustand, bei angewählten 16k Bereich 0 , im Bereich von 2048-8191 ($1000-$1FFF). Daß es zu Keinen Überschneidungen mit dem normalen RAM in diesem Bereich kommt, wird durch eine sehr ausgeklügelte Hardware erreicht, die die Systemtaktlücken ausnützt. Darauf werden wir hier jedoch nicht näher eingehen.



Beispiel der Grafikmöglichkeiten des Commodore 64. Der angewählte Zustand ist der STANDARD BIT MAP MODE. Die Grafik wurde mittels eines BASIC Programms berechnet und durch ein Maschinenprogramm in den Speicher geschrieben. Dieses Maschinenprogramm zum Einzelpunktsetzen wird noch besprochen.

3. Betriebsarten des Video Chips

Der Video Chip ist in der Lage in verschiedenen Betriebsarten zu arbeiten. Die einzelnen Zustände kann man durch Umprogrammieren der 47 Kontrollregister des Video Chips erreichen. Was dabei genau gemacht und beachtet werden muß, das besprechen wir in den nächsten Kapiteln ausführlich.
Die Betriebsarten sind im einzelnen:

- STANDARD CHARACTER MODE
  normale Zeichendarstellung eventuell mit umprogrammiertem Zeichensatz
- MULTI COLOR CHARACTER MODE
  verschiedenfarbige Zeichendarstellung
- EXTENDED BACKGROUND COLOR MODE
  verschiedene Hintergrundfarben bei normaler Zeichendarstellung (8 x 8 Matrix)
- STANDARD BIT MAP MODE
  hochauflösende Grafik
- MULTI COLOR BIT MAP MODE
  verschiedenfarbige hochauflösende Grafik
- SMOOTH SCROLLING
  sanftes Bildschirmverschieben
  in alle Richtungen
- STANDARD SPRITES
  freidefinierbare Figuren mit 24 x 21 Punkten
- MULTI COLOR SPRITES
  verschiedenfarbige Sprites
- SPRITEBETRIEBSARTEN
  Kollision der Sprites, Farbe der Sprites, ...
- SCREEN BLANKING
  Bildschirm ein/aus
- RASTER REGISTER
  hardwaremäßig gesteuertes Register
- INTERRUPT STATUS REGISTER
  interruptgesteuertes Register
- WEITERE REGISTER
  Light Pen, Interrupt Enable Register

3.1 Standard Character Mode

Wenn Sie Ihren Commodore 64 einschalten, dann befindet sich der Computer im sogenannten STANDARD CHARACTER MODE. Tippen Sie auf der Tastatur ein A, so erscheint es an der Stelle an der sich der Cursor gerade befindet. Ein weiteres Beispiel sind die reversen Buchstaben. Das reverse A erhalten Sie, indem Sie RVS ON (CTRL 9) tippen und dann das A eingeben. Soll aber anstelle der reversen Zeichen zum Beispiel das griechische Alphabet erscheinen, muß der Zeichensatz für die reversen Zeichen umprogrammiert werden. Das bedeutet, daß ein kleines Alpha erscheint, wenn Sie ein reverses A eingeben. Um einen eigenen Zeichensatz programmieren zu können, müssen wir zunächst einmal betrachten wie ein Zeichen abgespeichert ist. Ein Zeichen besteht aus 8 x 8 Einzelpunkten. Das bedeutet, daß wir insgesamt 64 Punkte abspeichern müssen um ein Zeichen eindeutig zu definieren. 64 Punkte entsprechen 64 Bit. 64 Bit sind wiederum 8 Byte. Wir brauchen also 8 Byte für ein Zeichen. Das erste Byte von diesen 8 Byte entspricht der ersten Zeile unseres Zeichens. Ist ein Bit auf 1, so ist auch der dazugehörige Punkt gesetzt. Ist das Bit 0, so nimmt der Punkt des Zeichens die Farbe des Hintergrundes an. Das bedeutet, daß sich der Punkt nicht vom übrigen Hintergrund unterscheidet. Die gesetzten Punkte nehmen im STANDARD CHARACTER MODE die Farbe, die gesetzt ist an. Sollen die Zeichen weiß sein, wird mittels CRTL 2 die Farbe weiß angewählt.
Auf der folgenden Seite ist der Aufbau eines einzelnen Zeichens noch einmal anschaulich dargestellt und es wird ein Beispiel für einen umgestalteten Zeichensatz gezeigt.

AUFBAU EINES ZEICHENS

BITMUSTER	AUFBAU								WERT DEZ.	HEX.
0000 0000									0	$00
0011 1000			*	*	*				43	$31
0110 1100		*	*		*	*			108	$6C
0110 1100		*	*		*	*			108	$6C
0110 1100		*	*		*	*			108	$6C
0111 1100		*	*	*	*	*			124	$7C
0110 1100		*	*		*	*			108	$6C
0000 0000									0	$00

Abbildung zum Aufbau eines einzelnen Zeichens. Unten ist ein veränderter Zeichensatz dargestellt. Als Beispiel wurde das kleine griechische Alphabet gewählt.

GRIECHISCHER ZEICHENSATZ

A	B	C	D	E	F	G	H	I	J	K	L	M
α	β	γ	δ	ε	ζ	η	ϑ	ι	χ	λ	μ	ν
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
ξ	o	π	ρ	σ	τ	υ	φ	χ	ψ	ω	↓	→

Wenn wir unseren eigenen Zeichensatz programmieren wollen, so müssen wir uns folgendes zuerst überlegen. Erstellen wir den neuen Zeichensatz ohne sofortige Kontrolle und schalten dann auf den neuen Satz um, oder verschieben wir den Commodore Standardzeichensatz und ändern ihn dann Zeichen für Zeichen um. Die zuletzt genannte Möglichkeit ist wohl die bessere, da wir sofort sehen können, wie unsere neugestalteten Zeichen aussehen. Vorher muß aber eine Position festgelegt werden, an der mit dem verschobenen Zeichensatz gearbeitet wird. Die beste Position, die sich dafür eignet, ist von Adresse 12288 ($3000) an aufwärts. Das hat folgenden Grund: befindet sich der Zeichensatz in niedrigeren Adressen, d.h. zum Beispiel bei Adresse 4096 ($1000), so würde ein etwas längeres BASIC Programm unseren neu erstellten Zeichensatz überschreiben und damit zerstören. Eine weitere denkbare Position wäre von Adresse 14336 ($3800) an aufwärts. Arbeiten wir aber mit einem vollen 4k Zeichensatz so ist diese Position ungeeignet, da wir dort 2k belegen dürfen. Deswegen wählen wir im folgenden für unseren verschobenen Zeichensatz den Bereich von 12288-16383 ($3000-$3FFF).
Zum Verändern der Zeichen eignet sich folgende Möglichkeit:

- Verschieben des Orginalzeichensatzes in den Bereich von 12288-16383 ($3000-$3FFF)
- Umschalten auf den verschobenen Zeichensatz
- Ändern bzw. Neugestaltung eigener Zeichen

Der Orginalzeichengenerator des Commodore 64 befindet sich ab Adresse 53248 ($D000). Dieser Bereich wird aber auch durch den Video Chip belegt. Durch einen Hardware-Trick ist es dennoch möglich, diese Überlappung zu realisieren. Wenn wir jetzt mit PEEK die zutreffenden Adressen auslesen, so erhalten wir die Inhalte der Register des Video Chips. Soll der Zeichengenerator ausles-

bar gemacht werden, so muß in der Adresse 1 ($01) das Bit 2 gelöscht werden. Dieses Bit bestimmt, ob der Video Chip angesprochen wird oder nicht. Würde man aber einfach den Video Chip ausschalten, so käme es zu Störungen im Computer. Aus diesem Grunde muß vorher der Interrupt ausgeschaltet werden, denn jedesmal wenn der Computer einen Interrupt durchläuft, wird hardwaremäßig der Video Chip angesprochen. Den Interrupt kontrolliert man mit Adresse 56334 ($DC0E). Ist Bit 0 gesetzt, so wird der Interrupt abgearbeitet, ist es gelöscht, so führt der Computer keinen Interrupt durch. Zusammenfassend ist noch einmal der eben besprochene Ablauf zur Umgestaltung des Zeichensatzes beschrieben.

- Ausschalten des Interrupts
- Ausschalten des Video Chips und damit den Zeichengenerator zum Auslesen freigeben
- Zeichensatz von Bereich 53248-57343 ($D000-$DFFF) nach Bereich 12288-16383 ($3000-$3FFF) verschieben
- Position auf neuen Zeichensatz setzen
- Video Chip einschalten
- Interrupt einschalten

BASIC:

```
POKE 56334,PEEK (56334) AND 254
; schaltet den Interrupt aus
```

MASCHINENSPRACHE:

```
    INTCTRL =$DC0E
LDA INTCTRL ; Wert laden
AND #$FE    ; Bit 0 löschen
STA INTCTRL ; schaltet den Interrupt aus
RTS
```

BASIC:

```
POKE 56334,PEEK (56334) OR 1
; schaltet den Interrupt ein
```

MASCHINENSPRACHE:

```
    INTCTRL =$DC0E
LDA INTCTRL ; Wert laden
ORA #$01    ; Bit 0 setzen
STA INTCTRL ; schaltet den Interrupt ein
RTS
```

BASIC:

```
POKE 1,PEEK (1) AND 251
; schaltet den Zeichengenerator auf Auslesen
  und den Video Chip aus
```

MASCHINENSPRACHE:

```
    VIDCTRL =$0001
LDA VIDCTRL ; Wert laden
AND #$FB    ; Bit 2 löschen
STA VIDCTRL ; schaltet den Zeichengenerator auf
RTS           Auslesen und den Video Chip aus
```

BASIC:

```
POKE 1,PEEK (1) OR 4
; schaltet den Video Chip ein und beendet das
  Auslesen des Zeichengenerators
```

MASCHINENSPRACHE:

```
    VIDCTRL =$0001
LDA VIDCTRL ; Wert laden
ORA #$04    ; Bit 2 setzen
STA VIDCTRL ; schaltet den Video Chip ein
RTS
```

BASIC:

```
POKE 53272,( PEEK (53272) AND 240) OR 12
; Zeichensatz 1 ab Adresse 12288 ($3000)
POKE 53272,( PEEK (53272) AND 240) OR 14
; Zeichensatz 2 ab Adresse 14336 ($3800)
```

MASCHINENSPRACHE:

```
    POS =$D018
LDA POS     ; Wert laden
AND #$F0    ; Bit 0...3 löschen
ORA #$0C    ; Bit 2, 3 setzen
STA POS     ; Zeichensatz 1 ab Adresse 12288
RTS           ($3000)

LDA POS     ; Wert laden
AND #$F0    ; Bit 0...3 löschen
ORA #$0E    ; Bit 1...3 setzen
STA POS     ; Zeichensatz 2 ab Adresse 14336
RTS           ($3800)
```

Fassen wir nun die oben einzeln aufgeführten Schritte in einem einzigen Programm zusammen.
Das Programm auf der nächsten Seite enthält alle erforderlichen Schritte, um den Zeichensatz zu verschieben und mit dem verschobenen Zeichensatz arbeiten zu können.

BASIC:

```
100 REM AUSLESEN DES ZEICHENSATZES
110 POKE56334,PEEK(56334)AND254
120 POKE1,PEEK(1)AND251
130 FORI=0TO4095
140 POKEI+12288,PEEK(I+53248)
150 NEXT
160 POKE1,PEEK(1)OR4
170 POKE56334,PEEK(56334)OR1
180 POKE53272,(PEEK(53272)AND240)OR12
190 END
```

Erläuterungen zum BASIC Programm:

110: Ausschalten des Interrupts
120: Ausschalten des Video Chips und damit Einschalten des Zeichengenerators zum Lesen
130: Schleife zum Verschieben des gesamten 4k Commodore Standardzeichensatzes
160: Einschalten des Video Chips
170: Einschalten des Interrupts
180: Position des Zeichengenerators auf 12288 ($3000) setzen

MASCHINENSPRACHE:

```
            INTCTRL =$DC0E
            VIDCTRL =$0001
            POS     =$D018
        LDA INTCTRL ; Wert der Interruptkontrolle
        AND #$FE    ; Bit 0 löschen
        STA INTCTRL ; schaltet den Interrupt aus
        LDA VIDCTRL ; Wert der Video Chip Kontrolle
        AND #$FB    ; Bit 2 löschen
        STA VIDCTRL ; Zeichensatz auf Auslesen
        LDA #$D0    ; H-Byte der Zeichensatzadresse
        STA $23     ; abspeichern in freie Adresse
        LDA #$30    ; H-Byte neue Position
        STA $25     ; abspeichern in freie Adresse
        LDA #$00    ; L-Byte beider Adressen
        STA $22     ; abspeichern
        STA $24     ; abspeichern
        LDX #$10    ; Zähler für die X-Schleife
XLOOP   LDY #$00    ; Zähler für die Y-Schleife
YLOOP   LDA ($22),Y ; Wert von alter Position laden
        STA ($24),Y ; auf neue Position abspeichern
        DEY         ; Y=Y-1
        BNE YLOOP   ; Y<>0 dann nach YLOOP springen
        INC $23     ; H-Byte Position alt +1
        INC $25     ; H-Byte Position neu +1
        DEX         ; X=X-1
        BNE XLOOP   ; X<>0 dann nach XLOOP springen
        LDA VIDCTRL ; Wert der Video Chip Kontrolle
        ORA #$04    ; Bit 2 setzen
        STA VIDCTRL ; Video Chip ein
        LDA INTCTRL ; Wert der Interruptkontrolle
        ORA #$01    ; Bit 0 setzen
        STA INTCTRL ; Interrupt ein
        LDA POS     ; Positionswert laden
        AND #$F0    ; Bit 0...3 löschen
        ORA #$0C    ; Position des Zeichengenerators
        STA POS     ; auf 12288 ($3000) setzen
        RTS
```

Arbeiten wir aber nun dennoch mit einem längeren BASIC Programm (ab etwa 8k), so empfiehlt es sich, den Speicherplatz zu begrenzen um damit der Zerstörung des verschobenen Zeichensatzes vorzubeugen.

BASIC:

```
POKE 56,48: CLR
; begrenzt den BASIC Speicherplatz
```

MASCHINENSPRACHE:

```
    MEM =$0038
    CLR =$A660
LDA #$30     ; Wert 48 ($30) laden
STA MEM      ; Speicherplatz begrenzen
JSR CLR      ; CLR ausführen
RTS
```

In Adresse 56 ($38) steht die höchste verfügbare RAM Adresse. Setzen wir diese Adresse auf 12288 ($3000), so kann unser verschobener Zeichensatz nicht mehr überschrieben werden.
Ab Adresse 42592 ($A660) steht im BASIC Interpreter des Commodore 64 die Routine für den BASIC Befehl CLR.

Ist eines der beiden Programme, also entweder das BASIC (Seite 26) oder das Maschinenprogramm (Seite 27) ausgeführt worden, so können wir nun die Zeichen verändern. Löschen Sie den Bildschirm mit CLR/HOME und tippen Sie in die linke obere Bildschirmecke einen Klammeraffen (Zeichen: @). Rechts daneben geben Sie ein A ein. Probieren Sie dann folgende Zeile aus:

BASIC:

```
POKE 12288,0: POKE 12298,0
```

MASCHINENSPRACHE:

```
LDA #$00      ; Wert 0 laden
STA $3000     ; erste Zeile aus dem @ löschen
STA $300A     ; dritte Zeile aus dem A loeschen
RTS
```

Wir stellen fest, daß beim Klammeraffen nun die obere Zeile und beim A die dritte Zeile fehlt. Jetzt können Sie Ihre eigenen Zeichen erstellen. Als Beispiel wollen wir anstelle des Klammeraffens folgendes Zeichen generieren:

NEUERSTELLTES ZEICHEN

BITMUSTER	AUFBAU								WERT DEZ.	HEX.
1111 1111	*	*	*	*	*	*	*	*	255	$FF
1000 0001	*							*	129	$81
1000 0001	*							*	129	$81
1001 1001	*			*	*			*	153	$99
1001 1001	*			*	*			*	153	$99
1000 0001	*							*	129	$81
1000 0001	*							*	129	$81
1111 1111	*	*	*	*	*	*	*	*	255	$FF

Ein kleines Programm, das auf der nächsten Seite abgedruckt ist, hilft uns bei der Erstellung dieses Zeichens. Der Einfachheit halber habe ich

mich bei diesem Beispiel auf ein einziges Zeichen beschränkt. Jederzeit können Sie das Programm für den kompletten Zeichensatz erweitern, indem Sie mittels einer Schleife Ihre neuen Werte der Zeichen einlesen.

BASIC:

```
100 REM NEUERSTELLEN DER ZEICHEN
110 FORI=0TO7
120 READA
130 POKEI+12288,A
140 NEXT
150 DATA255,129,129,153,153,129,129,255
160 END
```

MASCHINENSPRACHE:

```
LDA #$FF     ; Wert 255 ($FF) laden
STA $3000    ; 1. Zeile abspeichern
STA 3007     ; 8. Zeile abspeichern
LDA #$81     ; Wert 129 ($81) laden
STA $3001    ; 2. Zeile
STA $3002    ; 3. Zeile
STA $3005    ; 6. Zeile
STA $3006    ; 7. Zeile
LDA #$99     ; Wert 153 ($99) laden
STA $3003    ; 4. Zeile
STA $3004    ; 5. Zeile
RTS
```

Für diesen Fall ist wohl eindeutig das BASIC Programm das einfachere.

Folgendes Programm dient der Erstellung eines griechischen Zeichensatzes. In diesem Zusammenhang erübrigt sich ein Maschinenprogramm, da man sonst eine umständliche Tabelle generieren müßte.

BASIC:

```
100 REM GRIECHISCHER ZEICHENSATZ
110 FORI=0TO199
120 READA
130 POKEI+12288,A
140 NEXT
150 DATA056,108,108,108,056,000,000,000
160 DATA000,003,003,062,102,102,059,000
170 DATA056,108,108,124,102,102,124,096
180 DATA099,054,024,024,036,102,036,024
190 DATA060,096,096,060,102,102,060,000
200 DATA000,060,096,056,096,060,000,000
210 DATA056,014,024,048,096,111,051,006
220 DATA000,220,102,102,102,102,006,006
230 DATA028,054,031,006,198,102,102,060
240 DATA000,024,024,048,048,054,028,000
250 DATA000,230,172,056,056,108,198,000
260 DATA096,048,024,012,030,051,099,000
270 DATA000,102,102,102,102,127,096,096
280 DATA000,099,051,051,054,060,056,000
290 DATA096,060,096,124,096,096,126,006
300 DATA000,056,108,108,108,108,056,000
310 DATA000,031,118,054,054,054,055,000
320 DATA060,102,102,124,096,096,060,000
330 DATA000,030,056,108,102,102,060,000
340 DATA000,127,216,024,024,024,012,000
350 DATA000,108,230,102,102,102,060,000
360 DATA000,126,219,219,219,126,024,024
370 DATA099,150,028,024,024,053,102,000
380 DATA024,219,219,219,219,126,024,024
390 DATA000,000,219,219,219,219,126,000
```

3.2 Multi Color Character Mode

Der Multi Color Character Mode erlaubt es mehrfarbige Zeichen auf dem Bildschirm darzustellen. Im Gegensatz dazu kann im Standard Character Mode ein Zeichen nur eine Farbe für die gesamte 8 x 8 Matrix annehmen. Das ganze Zeichen ist dann z.B. weiß. Im Multi Color Character Mode kann ein einziges Zeichen insgesamt aus vier verschiedenen Farben bestehen, wobei eine Farbe der Hintergrundfarbe entspricht. Nimmt ein Punkt eines Zeichens die Farbe des Hintergrundes an, so ist es nicht sichtbar; es unterscheidet sich nicht vom übrigen Hintergrund. Die sich vom Hintergrund unterscheidenden Punkte können also drei verschiedene Farben haben.
Die Auflösung eines Zeichens halbiert sich im Multi Color Character Mode in X-Richtung. Das bedeutet, daß wir zwei Bit für einen Punkt benötigen. In X-Richtung, also horizontal, stehen uns somit für ein Zeichen vier Punkte, in Y-Richtung wie zuvor die 8 Zeilen zur Verfügung. Auf dem Bildschirm hingegen wird weiterhin die volle 8 x 8 Matrix angesprochen, nur mit der Einschränkung, daß eben immer zwei X-Einzelpunkte (Einzelpunkte die horizontal nebeneinander liegen) einen Doppelpunkt mit einer bestimmbaren Farbe bilden. Dies bietet einen erwünschten Nebeneffekt. Farbfernseher (Farbmonitore sind hiermit nicht angesprochen) würden die farbigen Einzelpunkte nur schlecht verarbeiten können.
Kontrolliert wird der Multi Color Character Mode durch Bit 4 des Video Chip Registers mit der Adresse 53270 ($D016). Ist das Bit 4 gesetzt, so ist der Multi Color Character Mode eingeschaltet. Ist es nicht gesetzt, so ist der Standard Character Mode eingeschaltet bzw. der Multi Color Character Mode ausgeschaltet. Zusätzlich besteht noch die Möglichkeit die beiden Character Modes zu mischen.

Ein- und Ausschalten des Multi Color Character Modes durch folgende Zeilen:

BASIC:

```
POKE 53270,PEEK (53270) OR 16
; schaltet den Multi Color Character Mode ein
```

MASCHINENSPRACHE:

```
    MCCM =$D016
LDA MCCM    ; Wert des Registers laden
ORA #$10    ; Bit 4 setzen
STA MCCM    ; Multi Color Character Mode ein
RTS
```

BASIC:

```
POKE 53270,PEEK (53270) AND 239
; schaltet den Multi Color Character Mode aus
```

MASCHINENSPRACHE:

```
    MCCM =$D016
LDA MCCM   ; Wert laden
AND #$EF   ; Bit 4 löschen
STA MCCM   ; Multi Color Character Mode aus
RTS
```

Den Multi Color Character Mode schaltet man generell mit der eben besprochenen Methode ein bzw. aus.
Des weiteren hat man für jede einzelne 8 x 8 Matrix auf dem Bildschirm die Möglichkeit, entweder den Standard Character Mode oder den Multi Color Character Mode anzuwählen.

Die Kontrolle darüber führt Bit 3 im Farb-RAM, also ab Adresse 55296 ($D800), aus. Und zwar für jede der 1000 8 x 8 Matrizen, die für die einzelnen Zeichen zuständig sind.
Nimmt ein Byte im Farb-RAM Werte von 0-7 ($00-$07) an (das entspricht den Farben schwarz, weiß, rot, erscheinen die Zeichen an den ensprechenden Stellen im Standard Character Mode. Das ergibt eine einfarbige Darstellung. Betragen aber die Werte im Farb-RAM 8-15 ($08-$0F), so ist das Bit 3 gesetzt, das bedeutet, daß die entsprechenden Zeichen im Multi Color Character Mode erscheinen.
Wie schon eingangs besprochen, bestimmen im Multi Color Character Mode immer zwei Bit einen Punkt. Diese zwei Bits können vier verschiedene Zustände annehmen:

Bits	Farbe des Punktes aus	Adresse
00	Hintergrundfarbregister 0	53281 ($D021)
01	Hintergrundfarbregister 1	53282 ($D022)
10	Hintergrundfarbregister 2	53283 ($D023)
11	Bit 0...2 des Farbrams	ab 55296 ($D800)

Besitzt ein Punkt die Farbe des Hintergrundfarbregisters 0, so ist er nicht sichtbar, denn er hat dieselbe Farbe wie der übrige Hintergrund.
Ein Zeichen besteht praktisch aus einer 4 x 8 programmierbaren Matrix. Als Beispiel sei hier die Erstellung eines verschiedenfarbigen A dargestellt. Die Hintergrndfarbe soll grau (Farbwert: 11 ($0B)) sein. Da unser A aus genau drei verschiedenen Farben bestehen kann, wählen wir weiß, rot und gelb. Um die Farben besser auf dem Papier unterscheiden zu können, setzen wir 1=weiß, 2=rot, 7=gelb und -=grau.

Das A würde dann so aussehen:

MEHRFARBIGES ZEICHEN

BITMUSTER	AUFBAU								WERT DEZ.	HEX.
0001 1000	-	-	1	1	2	2	-	-	24	$18
0011 1100	-	-	7	7	7	7	-	-	60	$3C
0110 0110	1	1	2	2	1	1	2	2	102	$66
0111 1110	1	1	7	7	7	7	2	2	126	$7E
0110 0110	1	1	2	2	1	1	2	2	102	$66
0110 0110	1	1	2	2	1	1	2	2	102	$66
0110 0110	1	1	2	2	1	1	2	2	102	$66
0000 0000	-	-	-	-	-	-	-	-	0	$00

Ein weiteres Beispiel ist das folgende Zeichen, das aus verschiedenfarbigen Zeilen besteht:

MEHRFARBIGES ZEICHEN

BITMUSTER	AUFBAU								WERT DEZ.	HEX.
0101 0101	1	1	1	1	1	1	1	1	85	$55
0101 0101	1	1	1	1	1	1	1	1	85	$55
1010 1010	2	2	2	2	2	2	2	2	170	$AA
1010 1010	2	2	2	2	2	2	2	2	170	$AA
0000 0000	-	-	-	-	-	-	-	-	0	$00
0000 0000	-	-	-	-	-	-	-	-	0	$00
1111 1111	7	7	7	7	7	7	7	7	255	$FF
1111 1111	7	7	7	7	7	7	7	7	255	$FF

BASIC:

```
100 REM VERSCHIEDENFARBIGE ZEICHEN
110 POKE56334,PEEK(56334)AND254
120 POKE1,PEEK(1)AND251
130 FORI=0TO2048
140 POKEI+12288,PEEK(I+53248)
150 NEXT
160 POKE1,PEEK(1)OR4
170 POKE56334,PEEK(56334)OR1
180 POKE53272,(PEEK(53272)AND240)OR12
190 FORI=0TO7
200 READA
210 POKEI+12288,A
220 NEXT
230 DATA85,85,170,170,0,0,255,255
240 PRINT"[illegible]";
250 POKE1024,0:POKE55296,15
260 POKE1026,0:POKE55298,7
270 POKE53281,11:POKE53282,1:POKE53283,2
280 POKE53270,PEEK(53270)OR16
290 END
```

Das aufgeführte Programm erstellt das auf der Seite 35 unten abgebildete mehrfarbige Zeichen und setzt es in die linke obere Bildschirmecke. Rechts daneben befindet sich dasselbe Zeichen nur einfarbig. Wegen der eventuell entstehenden Farbverwischungen durch die Wiedergabe über einen Farbfernseher wird die Erkennbarkeit des rechten Zeichens sehr geschwächt. Dieses Zeichen ist aber dennoch einfarbig. Das erkennen Sie, wenn Sie bei Ihrem Wiedergabegerät die Farbe wegnehmen. Sehr deutlich ist dann das Bitmuster 0101 0101 bzw. 1010 1010 zu erkennen. Die erste und zweite Zeile sind um einen Einzelpunkt gegenüber der dritten und vierten verschoben.
Das linke Zeichen besteht aus:
Zeile 1,2: weiß (Farbwert 1)
Zeile 3,4: rot (Farbwert 2)

Zeile 5,6: grau (Farbwert 11 = Hintergrundfarbe)
Zeile 7,8: gelb (Farbwert 7)

Erläuterungen zum BASIC Programm auf Seite 36:

110: Ausschalten des Interrupts
120: Video Chip aus, Zeichengenerator ein
130: Schleife zum Verschieben des Zeichensatzes
160: Video Chip ein
170: Einschalten des Interrupts
180: Position des Zeichengenerators auf 12288 ($3000) setzen
190: Schleife zum Neueinlesen des ersten Zeichens (Klammeraffe @ wird geändert)
240: Bildschirm löschen
250: Neuprogrammiertes Zeichen an linke obere Bildschirmecke setzen; Bit 3 der entsprechenden Position im Farb-RAM setzen; Bit 0...2 bestimmen die Farbe, wenn die beiden Bits zum Auswählen der Punktfarbe auf 1 sind; Farbe in unserem Fall 7, d.h. gelb (Bit 3 wird nicht mitgerechnet)
260: Neuprogrammiertes Zeichen mit Farbe gelb (dieses Zeichen ist einfarbig, weil Bit 3 nicht gesetzt ist) setzen
270: Beschreiben der Hintergrundfarbregister:
Register 0: grau (Farbwert 11)
Register 1: weiß (Farbwert 1)
Register 2: rot (Farbwert 2)
280: Multi Color Character Mode einschalten

MASCHINENSPRACHE:

```
INTCTRL =$DC0E
VIDCTRL =$0001
POS     =$D018
PRINT   =$FFD2
FREG0   =$D021
FREG1   =$D022
```

```
        FREG2 =$D023
        MCCM  =$D016
      LDA INTCTRL ; Interrupt Kontrollregister
      AND #$FE    ; Bit 0 löschen
      STA INTCTRL ; Interrupt aus
      LDA VIDCTRL ; Video Chip Kontrolle laden
      AND #$FB    ; Bit 2 löschen
      STA VIDCTRL ; Zeichensatz auf Auslesen
      LDA #$D0    ; H-Byte der Zeichensatzadresse
      STA $23     ; abspeichern in freie Adresse
      LDA #$30    ; H-Byte neue Position
      STA $25     ; abspeichern
      LDA #$00    ; L-Byte beider Adressen
      STA $22     ; abspeichern
      STA $24     ; abspeichern
      LDX #$10    ; Zähler für die X-Schleife
XLOOP LDY #$00    ; Zähler für die Y-Schleife
YLOOP LDA ($22),Y ; Wert von alter Position laden
      STA ($24),Y ; auf neue Position abspeichern
      DEY         ; Y=Y-1
      BNE YLOOP   ; Y<>0 dann nach YLOOP springen
      INC $23     ; H-Byte Position alt +1
      INC $25     ; H-Byte Position neu +1
      DEX         ; X=X-1
      BNE XLOOP   ; X<>0 dann nach XLOOP springen
      LDA VIDCTRL ; Wert der Video Chip Kontrolle
      ORA #$04    ; Bit 2 setzen
      STA VIDCTRL ; Video Chip ein
      LDA INTCTRL ; Wert der Interruptkontrolle
      ORA #$01    ; Bit 0 setzen
      STA INTCTRL ; Interrupt ein
      LDA POS     ; Positionswert laden
      AND #$F0    ; Bit 0...3 löschen
      ORA #$0C    ; Position des Zeichengenerators
      STA POS     ; auf 12288 ($3000) setzen
      LDA #$55    ; Wert 85 ($55) laden
      STA $3000   ; 1. Zeile des neuen Zeichens
      STA $3001   ; 2. Zeile
      LDA #$AA    ; Wert 170 ($AA) laden
      STA $3002   ; 3. Zeile
```

```
STA $3003    ; 4. Zeile
LDA #$00     ; Wert 0 ($00) laden
STA $3004    ; 5. Zeile
STA $3005    ; 6. Zeile
LDA #$FF     ; Wert 255 ($FF) laden
STA $3006    ; 7. Zeile
STA $3007    ; 8. Zeile
LDA #$93     ; ASCII Wert für CLR/HOME laden
JSR PRINT    ; CLR/HOME ausführen
LDA #$11     ; Wert für Cursor nach unten
JSR PRINT    ; 1x Cursor nach unten
JSR PRINT    ; 1x Cursor nach unten
LDA #$00     ; Code für @ laden
STA $0800    ; im Video-RAM abspeichern
STA $0802    ; im Video-RAM abspeichern
LDA #$0F     ; Farbwert 15 ($0F) laden
STA $D800    ; im Farb-RAM abspeichern
LDA #$07     ; Farbwert 7 ($07) laden
STA $D802    ; im Farb-RAM abspeichern
LDA #$0B     ; Farbwert 11 ($0B) grau laden
STA FREG0    ; im Farbregister 0 speichern
LDA #$01     ; Farbwert 1 ($01) weiß  laden
STA FREG1    ; im Farbregister 1 speichern
LDA #$02     ; Farbwert 2 ($02) rot laden
STA FREG2    ; im Farbregister 2 speichern
LDA MCCM     ; Wert laden
ORA #$10     ; Bit 4 setzen
STA MCMM     ; Multi Color Character Mode ein
RTS
```

Zum Auslesen des Zeichensatzes ist das Maschinenprogramm wesentlich schneller. Für den Rest des Programms, lohnt sich ein Maschinenprogramm nicht, da es nicht so sehr auf Schnelligkeit ankommt.
Das auf den vorangegangenen Seiten besprochene Programm stellt zwei Zeichen in der linken oberen Ecke des Bildschirms dar. Das linke der beiden ist ein Multi Color Character Mode Zeichen. Das bedeutet, daß das Zeichen aus mehreren Farben besteht, nämlich weiß, rot, grau und gelb.

3.3 Extended Background Color Mode

Der Extended Background Color Mode ist eine weitere Betriebsart des Video Chips. Er ermöglicht es, Zeichen mit verschiedenen Hintergrundfarben für jede einzelne 8 x 8 Matrix darzustellen. Somit können wir ein weißes Zeichen mit rotem Hintergrund (Hintergrund für die 8 x 8 Matrix) auf einem sonst grauen Bildschirm erzeugen. Die Farbauswahl wird durch die vier Hintergrundfarbregister getroffen. Eines der vier Register bestimmt die Hintergrundfarbe des gesamten Bildschirms, während die anderen drei Farbregister die Hintergrundfarben für eine 8 x 8 Matrix steuern. Das Farb-RAM hat dieselbe Aufgabe wie im Standard Character Mode, nämlich die Vordergrundfarbe eines Zeichens zu bestimmen. Die Auswahl, welches Zeichen welche Hintergrundfarbe bekommt, treffen Bit 6 und Bit 7 im Video-RAM (Video-RAM: RAM in dem die Werte der Zeichen auf dem Bildschirm stehen; im Einschaltzustand ab Adresse 1024 ($0400) bis 2023 ($07E7)). Wenn wir im Standard Character Mode an die linke obere Ecke ein A (Video-RAM Wert 1) setzen (POKE 1024,1), so erscheint das A mit der Vordergrundfarbe des Farb-RAM's (zum Beispiel weiß; Farb-RAM Wert 1; POKE 55296,1). Jetzt schreiben wir mit POKE an dieselbe Stelle im Video-RAM den Wert 65 und es erscheint das Zeichen SHIFT-A (CHR$(193)). Ist aber nun der Extended Background Color Mode eingeschaltet, so erscheint nicht das Zeichen SHIFT-A, jedoch das normale A mit der Hintergrundfarbe aus dem Farbregister 1. Wir können also im Extended Background Color Mode nur über 64 Zeichen verfügen, dafür aber mit insgesamt vier verschiedenen Hintergrundfarben für jedes einzelne Zeichen.
Eingeschaltet wird der Extended Background Color Mode durch Setzen des Bit 6 des Video Chip Kontrollregisters mit der Adresse 53265 ($D011).

BASIC:

```
POKE 53265,PEEK (53265) OR 64
; schaltet den Extended Background Color Mode ein
```

MASCHINENSPRACHE:

```
    EBCM =$D011
LDA EBCM     ; Wert laden
ORA #$40     ; Bit 6 setzen
STA EBCM     ; Extended Background Color Mode ein
RTS
```

BASIC:

```
POKE 53265,PEEK (53265) AND 191
; schaltet den Extended Background Color Mode aus
```

MASCHINENSPRACHE:

```
    EBCM =$D011
LDA EBCM     ; Wert laden
AND #$BF     ; Bit 6 löschen
STA EBCM     ; Extended Background Color Mode aus
RTS
```

Die folgende Tabelle gibt Aufschluß über Werte und Adressen im Extended Background Color Mode:

Video-RAM Wert	Bitmuster	Farbregister	Adresse
0 - 63	00XX XXXX	0	53281($D021)
64 - 127	01XX XXXX	1	53282($D022)
128 - 191	10XX XXXX	2	53283($D023)
192 - 255	11XX XXXX	3	53284($D024)

Die folgenden Programme in BASIC und Maschinensprache sollen ein Beispiel für die Benutzung des Extended Background Color Modes sein.

BASIC:

```
100 REM EXTENDED BACKGROUND COLOR MODE
110 POKE53280,2
120 POKE53281,12
130 POKE1024,9:POKE55296,1
140 POKE1025,73:POKE55297,1
150 POKE1026,137:POKE55298,1
160 POKE1027,201:POKE55299,1
170 POKE53282,0
180 POKE53283,2
190 POKE53284,7
200 POKE53265,PEEK(53265)OR64
210 PRINT"■"
220 END
```

Das Programm stellt in der linken oberen Ecke des Bildschirms viermal den Buchstaben I mit jeweils anderer Hintergrundfarbe dar.

Erläuterungen zum BASIC Programm:

110: Rahmenfarbe rot (Farbwert 2)
120: Hintergrundfarbe hellgrau (Farbwert 12)
130: Weißes I an die linke obere Bildschirmecke setzen; bei Video-RAM Wert 9 ist Bit 6 und Bit 7 nicht gesetzt, daraus folgt, daß die Farbe des Hintergrundes dieses Zeichens aus dem Farbregister 0 genommen wird.
140: Daneben wird ein weißes I gesetzt; hier beträgt der Video-RAM Wert 73 das bedeutet, daß Bit 6 gesetzt ist; die Hintergrundfarbe kommt aus dem Farbregister 1.

150: Der Video-RAM Wert beträgt hier 137, d.h. Bit 7 ist gesetzt; Farbe des Hintergrundes für dieses Zeichen kommt aus dem Hintergrundfarbregister 2.
160: Video-RAM Wert 201, d.h. Bit 6 und Bit 7 sind gesetzt, daher kommt die Farbe des Hintergrundes dieses Zeichens aus dem Hintergrundfarbregister 3.
170: Farbregister 1 auf schwarz, Farbwert 0 setzen.
180: Farbregister 2 auf rot, Farbwert 2 setzen.
190: Farbregister 3 auf gelb, Farbwert 7 setzen.
200: Extended Background Color Mode einschalten.
210: Der Kontrastwirkung wegen Schriftfarbe auf blau (Farbwert 6) setzen.

MASCHINENSPRACHE:

```
    PRINT  =$FFD2
    RAHMEN =$D020
    FREG0  =$D021
    FREG1  =$D022
    FREG2  =$D023
    FREG3  =$D024
    EBCM   =$D011
LDA #$93    ; ASCII Wert für  Bildschirm löschen
JSR PRINT   ; Bildschirm löschen
LDA #$02    ; Farbwert rot laden
STA RAHMEN  ; Rahmenfarbe ist rot
LDA #$0C    ; Farbwert hellgrau laden
STA FREG0   ; Hintergrundfarbe ist hellgrau
LDA #$09    ; Wert für  I mit 1. Hintergrundfarbe
STA $0400   ; links oben im Bildschirm setzen
LDA #$01    ; Vordergrundfarbe für  das I laden
STA $D800   ; abspeichern; I ist weiß
LDA #$49    ; Wert für  I mit 2. Hintergrundfarbe
STA $0401   ; daneben im Bildschirm abspeichern
LDA #$01    ; Vordergrundfarbe für  das I laden
STA $D801   ; abspeichern; I ist weiß
```

```
LDA #$89     ; Wert für  I mit 3. Hintergrundfarbe
STA $0402    ; im Bildschirm abspeichern
LDA #$01     ; Farbwert für  das I laden
STA $D802    ; I ist weiß
LDA #$C9     ; Wert für  I mit 4. Hintergrundfarbe
STA $0403    ; im Bildschirm abspeichern
LDA #$01     ; Farbwert für  das I laden
STA $D803    ; I ist weiß
LDA #$00     ; Farbe schwarz (Farbwert 0) laden
STA FREG1    ; im Farbregister 1 abspeichern
LDA #$02     ; Farbe rot (Farbwert 2) laden
STA FREG2    ; im Farbregister 2 abspeichern
LDA #$07     ; Farbe gelb (Farbwert 7) laden
STA FREG3    ; im Farbregister 3 abspeichern
LDA EBCM     ; Wert laden
ORA #$40     ; Bit 6 setzen
STA EBCM     ; Extended Background Color Mode ein
LDA #$1F     ; ASCII Wert für  Farbe blau laden
JSR PRINT    ; wegen Kontrast auf blau umschalten
LDA #$11     ; Wert für  Cursor nach unten laden
JSR PRINT    ; 1x Cursor nach unten ausführen
RTS
```

Die beiden Programme führen exakt dasselbe aus. Auch hier bringt das Maschinenprogramm kaum Vorteile, da es bei diesem Beispiel nicht so sehr auf die Geschwindigkeit ankommt, mit der das Programm ausgeführt wird.

4. Hochauflösende Grafiken

Das wohl interessanteste Kapitel beim Commodore 64 ist eindeutig die hochauflösende Grafik. Leider verrät das Commodore-64-Bedienungshandbuch, das beim Kauf des Computers mitgeliefert wird, kein einziges Wort darüber. Mit der hochauflösenden Grafik hat man die Möglichkeit über 64000 Einzelpunkte zu verfügen. Jeden dieser Einzelpunkte kann man gesondert ansprechen. Zusammen mit etwas Mathematik, entsteht dann eine solche dreidimensionale Grafik, wie sie auf der nächsten Seite abgebildet ist. Die 64000 Einzelpunkte gliedern sich in 320 x 200 Punkte auf. Das bedeutet, daß uns in X-Richtung (horizontal) insgesamt 320 Punkte und in Y-Richtung (vertikal) 200 Zeilen zur Verfügung stehen. Der große Nachteil ist die umständliche Ansteuerung der Grafik, um beispielsweise einen Punkt zu setzen. Andere Computer, wie zum Beispiel der Apple, gestatten von Haus aus mit einfachen Befehlen Punkte zu setzen oder Linien zu zeichnen. Um aber doch die Grafikansteuerung so komfortabel wie möglich zu gestalten, wurden diverse, zum Teil schon hinreichend bekannte, Hilfsprogramme erstellt. BASIC Programme dienen zwar dem Verständnis, keinesfalls aber der Schnelligkeit, um zum Beispiel eine Linie zu zeichnen. Für diese komplizierten Aufgaben eignet sich zweifellos ein Maschinenprogramm am besten. Nun ist es aber nicht jedermanns Sache, schwierige Probleme in Maschinensprache zu programmieren. Aus diesem Grund finden Sie in diesem Buch BASIC- und Maschinen-Programme nebeneinander.
Für die schnelle Grafikansteuerung wurde auch der IWT SPRITE KOMFORT KIT entwickelt. Dieses 4k lange Maschinenprogramm enthält cirka 40 neue Befehle, die nicht nur die Grafikansteuerung erleichtern. Eine Liste der Befehle befindet im Anhang dieses Buches.

Beispiel einer hoch-
auflösenden Grafik
im sogenannten
STANDARD BIT MAP
MODE.

4.1 Standard Bit Map Mode

Der Standard Bit Map Mode ist die Betriebsart des Video Chips, mit der man über die Einzelpunkte des gesamten Bildschirms verfügen kann. Wie schon im Kapitel 'hochauflösende Grafiken' erwähnt, sind es insgesamt 64000, also 320 x 200 Einzelpunkte. Ein einzelner Punkt kann zwei Zustände annehmen: gesetzt oder nicht gesetzt. Zur Ansteuerung eines Punktes eignet sich somit ein Bit, das ebenfalls nur zwei Zustände annehmen kann, nämlich 1 oder 0. Um nun alle 64000 Punkte ansprechen zu können, benötigen wir insgesamt 64000 Bit. 64000 Bit entsprechen 8000 Byte (fast 8k). Wir müssen also für den hochauflösenden Bildschirm 8000 Byte zur Verfügung stellen.
Die Farbe können wir im Standard Bit Map Mode nur für eine 8 x 8 große Gruppe von Punkten bestimmen. Und zwar einmal die Farbe der Punkte in der Gruppe, die gesetzt sind. Zum anderen die Farbe der Punkte, die nicht gesetzt sind, die Hintergrundfarbe. Einzige Besonderheit: die Farbwerte stehen nicht wie gewohnt im Farb-RAM, sondern im Video-RAM (normal von 1024 ($0400) an aufwärts). Aber dazu später mehr.
Ein- bzw. Ausgeschaltet wird der Standard Bit Map Mode durch Bit 5 der Adresse 53265 ($D011):

BASIC:

```
POKE 53265,PEEK (53265) OR 32
; Standard Bit Map Mode ein
```

MASCHINENSPRACHE:

```
    SBMM =$D011
LDA SOMM     ; Wert laden
ORA #$20     ; Bit 5 setzen
```

```
STA SBMM    ; Standard Bit Map Mode ein
RTS
```

BASIC:

```
POKE 53265,PEEK (53265) AND 223
; Standard Bit Map Mode aus
```

MASCHINENSPRACHE:

```
    SBMM =$D011
LDA SBMM    ; Wert laden
AND #$DF    ; Bit 5 löschen
STA SBMM    ; Standard Bit Map Mode aus
RTS
```

Die Farbdefinition der gesetzten Punkte bzw. der nicht gesetzten Punkte erfolgt etwas anders als gewohnt. Die unteren vier Bits, Bit 0...3, bestimmen die Farbe der nicht gesetzten Punkte. Die oberen vier Bits, Bit 4...7, bestimmen die Farbe der gesetzten Punkte. Möchten wir beispielsweise weiße Punkte auf rotem Hintergrund haben, so sieht unser Bitmuster wie folgt aus:

```
0001  0010 = 18 ($12)
weiß  rot
```

Nochmals sei darauf hingewiesen, daß die Farbdefinition des Standard Bit Map Modes nicht im Farb-RAM, sondern im Video-RAM steht.
Ein Problem gibt es noch zu lösen, und zwar die Position der Bit Map, also des 8000 Byte Speicherbereiches, der kontrolliert, ob die Punkte gesetzt sind oder nicht.
Wir wählen den Bereich von 8192-16191 ($2000-$3F3F). In diesem Bereich müssen wir uns nicht um die 16k Auswahl kümmern (vergleiche Kapitel 2.2).

Zusammenfassend müssen wir folgende Schritte bearbeiten, um mit dem Standard Bit Map Mode arbeiten zu können:

- Bit Map Position an Adresse 8192-16191 ($2000-$3F3F)
- Bit Map löschen
- Farbe der Punkte bzw. Hintergrundfarbe setzen
- Standard Bit Map Mode einschalten

Nachfolgend werden die eben aufgeführten Schritte anhand von Programmzeilen aufgezeigt:

BASIC:

```
POKE 53272,PEEK (53272) OR 8
; Position der Bit Map auf Startadresse 8192
  ($2000) setzen
```

MASCHINENSPRACHE:

```
    POS =$D018
LDA POS       ; Positionswert laden
ORA #$08      ; Bit 3 setzen
STA POS       ; Bit Map ab 8192 ($2000)
RTS
```

BASIC:

```
FOR I=0 TO 7999
POKE I+8192,0
NEXT          ; Bit Map löschen
```

MASCHINENSPRACHE:

```
LDA #$20      ; H-Byte der Startadresse der Bit Map
```

```
      STA $25       ; in freie Adresse abspeichern
      LDA #$00      ; L-Byte der Startadresse laden
      STA $24       ; abspeichern
      LDX #$20      ; Zähler für X-Schleife
XLOOP LDY #$00      ; Zähler für Y-Schleife
YLOOP STA ($24),Y ; Akku=0 abspeichern
      INY           ; Y=Y+1
      BNE YLOOP     ; Y<>0 dann nach YLOOP springen
      INC $25       ; H-Byte +1
      DEX           ; X=X-1
      BNE XLOOP     ; X<>0 dann nach XLOOP springen
      RTS
```

BASIC:

```
FOR I=1024 TO 2023 ; Schleife für die Farbbestim-
POKE I,27            mung; hier: 1*16+11=27
NEXT                 weiß (Farbwert 1) auf grau
                     (Farbwert 11)
```

MASCHINENSPRACHE:

```
      LDA #$1B      ; Wert für  weiß  auf grau laden
      LDX #$00      ; X-Zähler
LOOP  STA $0400,X ; das gesamte Video-RAM wird
      STA $0500,X ; mit dem Wert 27 ($1B) be-
      STA $0600,X ; schrieben; gesetzte Punkte
      STA $0700,X ; haben alle die gleiche Farbe
      INX           ; X=X+1
      BNE LOOP      ; X<>0 dann nach LOOP springen
      RTS
```

BASIC:

```
POKE 53265,PEEK (53265) OR 32
; Standard Bit Map Mode ein
```

MASCHINENSPRACHE:

```
    SBMM =$D011
LDA SBMM    ; Wert laden
ORA #$20    ; Bit 5 setzen
STA SBMM    ; Standard Bit Map Mode ein
RTS
```

BASIC:

```
POKE 8192,128
POKE 8193,64
POKE 8194,16 ; Punkte im hochauflösenden  Grafik-
POKE 8195,8  ; bildschirm setzen
```

MASCHINENSPRACHE:

```
LDA #$80    ; Wert für  die 1. Zeile
STA $2000   ; in die Bit Map abspeichern
LDA #$40    ; Wert für  die 2. Zeile
STA $2001   ; abspeichern
LDA #$10    ; Wert für  die 3. Zeile
STA $2002   ; abspeichern
LDA #$08    ; Wert für  die 4. Zeile
STA $2003   ; abspeichern
RTS
```

Aufbau der Bit Map ab Adresse 8192 ($2000):

```
8192 XXXXXXXX  8200 XXXXXXXX  8208 XXXXXXXX ...
8193 XXXXXXXX  8201 XXXXXXXX  8209 XXXXXXXX ...
8194 XXXXXXXX  8202 XXXXXXXX  8210 XXXXXXXX ...
8195 XXXXXXXX  8203 XXXXXXXX  8211 XXXXXXXX ...
8196 XXXXXXXX  8204 XXXXXXXX  8212 XXXXXXXX ...
8197 XXXXXXXX  8205 XXXXXXXX  8213 XXXXXXXX ...
8198 XXXXXXXX  8206 XXXXXXXX  8214 XXXXXXXX ...
8199 XXXXXXXX  8207 XXXXXXXX  8215 XXXXXXXX ...
```

Fassen wir nun die einzeln besprochenen Programmteile in eimen kompletten Programm zusammen. Zuvor sei noch gesagt, daß im Zusammenhang mit der hochauflösenden Grafik Maschinenprogramme den BASIC Programmen an Schnelligkeit weit überlegen sind. Warten Sie also beim BASIC Programm ruhig etwas länger. Das Maschinenprogramm bringt es in Bruchteilen einer Sekunde.

BASIC:

```
100 REM STANDARD BIT MAP MODE
110 POKE53272,PEEK(53272)OR8
120 POKE53265,PEEK(53265)OR32
130 FORI=0TO7999
140 POKEI+8192,0
150 NEXT
160 FORI=1024TO2023
170 POKEI,27
180 NEXT
190 POKE8192,128
200 POKE8193,64
210 POKE8144,16
220 POKE8195,8
230 IFPEEK(203)=64THEN230
240 POKE53265,PEEK(53265)AND223
250 POKE53272,PEEK(53272)AND247
260 PRINT"♥";:POKE198,0
270 END
```

Erläuterungen zum BASIC Programm:

110: Setzen der Bit Map Position auf Startadresse 8192 ($2000)
120: Standard Bit Map Mode einschalten
130: Schleife zum Löschen des hochauflösenden Grafikbildschirms
160: Schleife zum Setzen der Farbinformation; hier: weiß auf grau (Farbwert 1 auf

Farbwert 11)
190: Setzen verschiedener Einzelpunkte auf dem Grafikbildschirm
230: Warten auf Tastendruck
240: Standard Bit Map Mode aus
250: Position wieder auf den Standardwert zurücksetzen, damit wieder mit dem normalen Bildschirm gearbeitet werden kann
260: Bildschirm löschen und Anzahl der gedrückten Tasten auf 0 setzen, damit nicht im Bildschirm das eben gedrückte Zeichen erscheint

MASCHINENSPRACHE:

```
          POS    =$D018
          SBMM   =$D011
          TASTE  =$00CB
          PRINT  =$FFD2
          ANZTAS =$00C6
      LDA POS      ; Positionswert laden
      ORA #$08     ; Bit 3 setzen
      STA POS      ; auf 8192 ($2000) setzen
      LDA SBMM     ; Wert laden
      ORA #$20     ; Bit 5 setzen
      STA SBMM     ; Standard Bit Map Mode ein
      LDA #$20     ; H-Byte der Startadresse laden
      STA $25      ; in freie Adresse abspeichern
      LDA #$00     ; L-Byte der Startadresse laden
      STA $24      ; abspeichern
      LDX #$20     ; X-Schleifenzähler
XLOO  LDY #$00     ; Y-Schleifenzähler
YLOOP STA ($24),Y  ; Akku=0 abspeichern
      DEY          ; Y=Y-1
      BNE YLOOP    ; Y<>0 dann nach YLOOP springen
      INC $25      ; H-Byte des Zählers  +1
      DEX          ; X=X-1
      BNE XLOOP    ; X<>0 dann nach XLOOP springen
      LDA #$1B     ; Wert für  weiß  auf grau
```

```
      LDX #$00      ; X-Schleifenzähler
LOOP  STA $0400,X   ; das gesamte Video-RAM wird
      STA $0500,X   ; mit dem gleichen Farbwert
      STA $0600,X   ; 27 ($1B) beschrieben; das
      STA $0700,X   ; entspricht weiß  auf grau
      INX           ; X=X+1
      BNE LOOP      ; X<>0 dann nach LOOP springen
      LDA #$80      ; Wert für  die 1. Zeile laden
      STA $2000     ; in die Bit Map abspeichern
      LDA #$40      ; Wert für  die 2. Zeile laden
      STA $2001     ; abspeichern
      LDA #$10      ; Wert für  die 3. Zeile laden
      STA $2002     ; abspeichern
      LDA #$08      ; Wert für  die 4. Zeile laden
      STA $2003     ; abspeichern
TLOOP LDA TASTE     ; Wert der gedrückten  Taste
      CMP #$40      ; keine Taste gedrückt?
      BEQ TLOOP     ; ja, dann nach TLOOP springen
      LDA SBMM      ; Wert laden
      AND #$DF      ; Bit 5 löschen
      STA SBMM      ; Standard Bit Map Mode aus
      LDA POS       ; Positionswert laden
      AND #$F7      ; Bit 3 löschen
      STA POS       ; Position auf Standardwert
      LDA #$93      ; ASCII Wert für  CLR/HOME
      JSR PRINT     ; Bildschirm löschen
      LDA #$00      ; Anzahl der gedrückten
      STA ANZTAS    ; Tasten = 0
      RTS
```

Die Programme (BASIC oder Maschinensprache) setzen verschiedene Punkte in die linke obere Ecke des hochauflösenden Grafikbildschirms. Sie können selbst beliebige Speicherstellen im Bereich von 8192-16191 ($2000-$3F3F) beschreiben. Zum Ausprobieren eignet sich besser das BASIC Programm. Folgende Zeile, die sich ins BASIC Programm leicht einfügen läßt, setzt andere Punkte:

```
255 POKE 10000,255
```

Uns stehen in X-Richtung 320 Punkte und in Y-Richtung 200 Punkte zur Verfügung. Was liegt da näher als die Punkte mittels Koordinaten anzusprechen und somit den Ansteuerungskomfort zu erhöhen. Suchen wir also eine Lösung, damit wir über eine derartige Eingabe verfügen können:

```
Eingabe : X = 230 : Y = 50
Ergebnis: Punkt mit den Koordinaten 230,50 wird
          gesetzt
```

Um eine solche Eingabe zu erreichen, müssen mehrere Berechnungen angestellt werden. Wollen wir einen Punkt setzen, so ist als erstes die Reihe der 8 x 8 Matrix, in der sich der Punkt befindet, zu errechnen:

```
RE = INT (Y/8) ; Reihe berechnen
```

Die Reihe der 8 x 8 Matrix errechnet sich aus dem Y Punktwert. Dasselbe gilt auch für den X Punktwert. Die Spalte läßt sich demnach aus dem X Wert berechnen:

```
SP = INT (X/8) ; Spalte berechnen
```

Reihe und Spalte sind sozusagen die Koordinaten der 8 x 8 Matrix. RE kann Werte von 0...24 annehmen, SP Werte von 0...39. Denn unser Bildschirm besteht ja, wenn man ihn in 8 x 8 Gruppen zusammenfaßt aus 25 Zeilen mit je 40 Zeichen.
Nachdem die 8 x 8 Gruppe berechnet ist, müssen wir nur noch innerhalb dieser Gruppe Berechnungen anstellen. Zu berechnen ist, welche Zeile der insgesamt 8 Zeilen diejenige ist, in der sich das zu ändernde Bit (der zu setzende Punkt) befindet:

```
ZE = Y AND 7    ; berechnet die Zeile innerhalb
                  einer 8 x 8 Gruppe
```

Mit dieser Zeile haben wir nun das Byte gefunden, das wir um ein zu setzendes Bit ändern. Dieses Bit erhalten wir mit:

```
BI = 7 - (X AND 7)
```

Nachdem jetzt alle Einzeldaten unseres Bytes und Bits berechnet sind, fehlt uns nur noch die Adresse des Bytes. Da unsere Bit Map bei Adresse 8192 ($2000) beginnt, addiert sich diese Anfangsadresse dazu:

```
BY = 8192 + RE * 320 + SP * 8 + ZE
```

Nun setzen wir den entsprechenden Punkt mittels eines POKE Befehls:

```
POKE BY,PEEK (BY) OR 2 ↑ BI
```

Benutzen wir also folgende BASIC Zeilen um einen Punkt zu setzen:

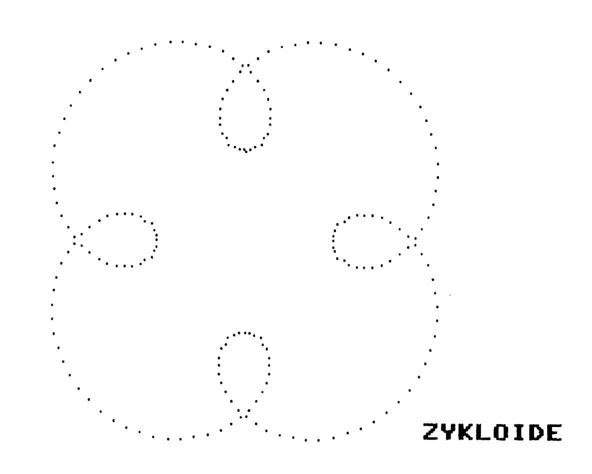
```
X = 0...319 : Y = 0...199     ; X,Y Wert einlesen
RE = INT (Y/8)                ; Reihe berechnen
SP = INT (X/8)                ; Spalte berechnen
ZE = Y AND 7                  ; Zeile berechnen
BI = 7 - (X AND 7)            ; Bit berechnen
BY = 8192 + RE*320 + SP*8 +ZE ; Byte berechnen
POKE BY , PEEK (BY) OR 2 ↑ BI ; Bit setzen
```

Diese Programmzeilen schreibt man am besten als Unterprogramm irgendwo am Ende des Hauptproramms. Soll dann ein Punkt gesetzt werden, springt man einfach mit den gesetzten Variabeln X und Y in das Unterprogramm.
Die eben besprochene Punktsetztroutine soll am Beispiel einer Zykloide (eine Zykloide ist eine mathematische Relation, bestehend aus Sinus- und Kosinusfunktionen) demonstriert werden:

BASIC:

```
100 REM ZYKLOIDE
110 POKE53272,PEEK(53272)OR8
120 POKE53265,PEEK(53265)OR32
130 FORI=0TO7999
140 POKEI+8192,0
150 NEXT
160 FORI=1024TO2023
170 POKEI,27
180 NEXT
190 FORI=0TO2*πSTEP.0314159265
200 X=160-70*SIN(I)+30*SIN(5*I)
210 Y=100-70*COS(I)+30*COS(5*I)
220 GOSUB1000
230 NEXT
240 IFPEEK(203)=64THEN240
250 POKE53265,PEEK(53265)AND223
260 POKE53272,PEEK(53272)AND247
270 PRINT"♥";:POKE198,0
280 END
1000 REM PUNKT SETZEN
1010 RE=INT(Y/8)
1020 SP=INT(X/8)
1030 ZE=YAND7
1040 BI=7-(XAND7)
1050 BY=8192+RE*320+SP*8+ZE
1060 POKEBY,PEEK(BY)OR2↑BI
1070 RETURN
```

Dieses Programm zeichnet eine Zykloide in hochauflösender Grafik. Ist die Grafik erstellt, wartet der Computer auf einen beliebigen Tastendruck und beendet dann den Standard Bit Map Mode. Die erstellte Grafik ist auf der nächsten Seite abgebildet.



Erläuterungen zum BASIC Programm:

110: Setzen der Bit Map Position auf die Startadresse 8192 ($2000)
120: Standard Bit Map Mode einschalten
130: Schleife zum Löschen des hochauflösenden Grafikbildschirms
160: Schleife zum Setzen der Farbinformation hier: weiß auf grau (Farbwert 1 auf Farbwert 11)
190: Schleife für die mathematische Berechnung der zu setzenden Einzelpunkte der Zykloide
220: Sprung in die als Unterprogramm gestaltete Punktsetzroutine
240: Warten auf Tastendruck
250: Standard Bit Map Mode aus
260: Position wieder auf den Standardwert zurücksetzen, damit mit dem normalen Bildschirm gearbeitet werden kann

```
1000: Beginn des Punksetzunterprogramms
1010: Reihe berechnen
1020: Spalte berechnen
1030: Zeile berechnen
1040: Bit berechnen
1050: Byte berechnen
1060: Punkte setzen
1070: Rücksprung aus dem Unterprogramm
```

Das aufgezeigte BASIC Programm benötigt leider recht lange, bis das gewünschte Bild fertig ist. So stellt sich schon bald die Frage, wie man diesen Vorgang in Maschinensprache beschleunigen kann. Wir wollen jetzt aber nicht besprechen, wie man die Zeilen mit Sinus und Kosinus in Maschinensprache 'übersetzt'. Vielmehr wollen wir uns hier damit befassen, ein Maschinenprogramm zu schreiben, das einen Punkt im hochauflösenden Grafikbildschirm setzt. Das Programm soll etwa folgenden Befehlsmodus verstehen:

```
SYS 49152,X,Y ; setzt Punkt an die Stelle X,Y
```

Das Programm, das nachfolgend aufgeführt ist, bewirkt diese Forderung:

MASCHINENSPRACHE:

```
        CHRGET =$0073
        KOLES  =$B7EB
    JSR CHRGET  ; nächstes  Zeichen holen
    JSR KOLES   ; Koordinatenwerte einlesen
    STX $23     ; Y Koordinate abspeichern
    LDA $14     ; L-Byte der X-Koordinate laden
    AND #$07    ; Bits 3...7 löschen
    TAX         ; Akku nach X bringen
    EOR $14     ; Akku EOR Adresse $14
    STA $14     ; abspeichern
    LDA #$00    ; 0 laden
```

```
      STA $22       ; abspeichern
      SEC           ; Carry Bit setzen (C=1)
LOOP  ROR           ; Akku um ein Bit nach rechts
      DEX           ; X=X-1
      BPL LOOP      ; X=255?; nein, dann nach LOOP
      TAX           ; A nach X bringen
      LDA $23       ; Y Koordinate des Punktes
      AND #$07      ; Bits 3...7 löschen
      TAY           ; A nach Y bringen
      EOR $23       ; A EOR Adresse $23
      STA $23       ; abspeichern
      LSR           ; A um zwei Bits nach rechts
      LSR           ; verschieben
      ADC $23       ; Adresse $23 addieren
      LSR           ; A um zwei Bits nach rechts
      LSR           ; verschieben
      ROR $22       ; Adr. $22 ein Bit nach rechts
      LSR           ; A ein Bit nach rechts
      ROR $22       ; Adr. $22 ein Bit nach rechts
      STA $23       ; abspeichern
      LDA $14       ; Wert in Adresse $14 laden
      ADC $22       ; Adr. $22 addieren
      STA $22       ; abspeichern
      LDA $15       ; Wert in Adresse $15 laden
      ADC $23       ; Adr. $23 addieren
      ORA #$20      ; Bit 5 setzen
      STA $23       ; abspeichern
      TXA           ; X nach A bringen
      ORA ($22),Y   ; entsprechende Bits setzen
      STA ($22),Y   ; Punkte setzen
      RTS
```

Bevor wir nun zu einer Zusammenfassung und der Anwendung der Einzelprogramme kommen, sei hier noch kurz aufgezeigt, wie man einen Punkt wieder löschen kann.
Schauen wir uns einmal das Programm auf Seite 56 an. Diese Routine in BASIC setzt einen Punkt. Interessant für uns ist dabei nur die letzte Zeile, denn die vorangehende Berechnung ist auch

zum Löschen eines Punktes nötig. Verändern wir die Zeile wie folgt, so können wir auch einen gesetzten Punkt wieder löschen:

BASIC:

```
POKE BY,PEEK (BY) OR 2 ↑ BI            ; setzen

POKE BY,PEEK (BY) AND (255 - 2 ↑ BI) ; löschen
```

Wir müssen nur den Rest der Zeile OR 2 ↑ BI durch AND (255 - 2 ↑ BI) ersetzen, und können bereits gesetzte Punkte löschen. Ein Beispiel folgt später.
Dasselbe können wir natürlich auch in Maschinensprache erreichen. Dort gilt unsere Aufmerksamkeit dem Maschinenprogramm auf Seite 60 und zwar den letzten drei Zeilen:

MASCHINENSPRACHE:

```
ORA ($22),Y  ; entsprechende Bits setzen
STA ($22),Y  ; Punkt setzen

EOR $#FF     ; Bit 0...7 negieren
AND ($22),Y  ; entsprechende Bits löschen
STA ($22),Y  ; Punkt löschen
```

Eine Zeile, nämlich EOR #$FF, muß neu eingefügt und die darauffolgende Zeile geändert werden: AND ($22),Y.
Wir werden nun die Einzelprogramme zu einem Maschinenprogramm zusammenfassen, das z.B. ab Adresse 49152 ($C000) im Speicher liegt. Dieses Hilfsprogramm dient auch zum Löschen des Grafikbildschirms und zum Setzen des Farbspeichers.

Hier verwenden wir erstmals ein gemischtes Programm. Das bedeutet, daß das BASIC Programm ein Maschinenprogramm aufruft. Das hier gezeigte Maschinenprogramm werden wir auch später öfters mit anderen Programmen verwenden.

MASCHINENSPRACHE:

```
C000 AD 18 D0    LDA $D018
C003 09 08       ORA #$08
C005 8D 18 D0    STA $D018
C008 AD 11 D0    LDA $D011
C00B 09 20       ORA #$20
C00D 8D 11 D0    STA $D011
C010 60          RTS
C011 A9 20       LDA #$20
C013 85 25       STA $25
C015 A9 00       LDA #$00
C017 85 24       STA $24
C019 A2 20       LDX #$20
C01B A0 00       LDY #$00
C01D 91 24       STA ($24),Y
C01F 88          DEY
C020 D0 FB       BNE $C01D
C022 E6 25       INC $25
C024 CA          DEX
C025 D0 F4       BNE $C01B
C027 60          RTS
C028 A9 1B       LDA #$1B
C02A A2 00       LDX #$00
C02C 9D 00 04    STA $0400,X
C02F 9D 00 05    STA $0500,X
C032 9D 00 06    STA $0600,X
C035 9D 00 07    STA $0700,X
C038 E8          INX
```

```
C039 D0 F1       BNE $C02C
C03B 60          RTS
C03C 20 4E C0    JSR $C04E
C03F 11 22       ORA ($22),Y
C041 91 22       STA ($22),Y
C043 60          RTS
C044 20 4E C0    JSR $C04E
C047 49 FF       EOR #$FF
C049 31 22       AND ($22),Y
C04B 91 22       STA ($22),Y
C04D 60          RTS
C04E 20 73 00    JSR $0073
C051 20 EB B7    JSR $B7EB
C054 86 23       STX $23
C056 A5 14       LDA $14
C058 29 07       AND #$07
C05A AA          TAX
C05B 45 14       EOR $14
C05D 85 14       STA $14
C05F A9 00       LDA #$00
C061 85 22       STA $22
C063 38          SEC
C064 6A          ROR
C065 CA          DEX
C066 10 FC       BPL $C064
C068 AA          TAX
C069 A5 23       LDA $23
C06B 29 07       AND #$07
C06D A8          TAY
C06E 45 23       EOR $23
C070 85 23       STA $23
C072 4A          LSR
C073 4A          LSR
C074 65 23       ADC $23
C076 4A          LSR
C077 4A          LSR
C078 66 22       ROR $22
C07A 4A          LSR
C07B 66 22       ROR $22
C07D 85 23       STA $23
C07F A5 14       LDA $14
```

```
C081 65 22        ADC $22
C083 85 22        STA $22
C085 A5 15        LDA $15
C087 65 23        ADC $23
C089 09 20        ORA #$20
C08B 85 23        STA $23
C08D 8A           TXA
C08E 60           RTS
C08F AD 11 D0     LDA $D011
C092 29 DF        AND #$DF
C094 8D 11 D0     STA $D011
C097 AD 18 D0     LDA $D018
C09A 29 F7        AND #$F7
C09C 8D 18 D0     STA $D018
C09F A9 93        LDA #$93
C0A1 20 D2 FF     JSR $FFD2
C0A4 A9 00        LDA #$00
C0A6 85 C6        STA $C6
C0A8 60           RTS
```

Erläuterungen zum Maschinenprogramm:

C005: Position der Bit Map auf 8192 ($2000) setzen
C00D: Standard Bit Map Mode ein
C017: Werte für die Schleife einlesen
C025: Schleife zum Löschen der Bit Map
C028: Farbwert einlesen; hier: weiß auf grau (Farbwert 1 auf Farbwert 11)
C039: Schleife zum Beschreiben mit der Farbinformation
C03C: Position für den Punkt berechnen
C041: Punkt setzen
C044: Position für den Punkt berechnen
C04B: Punkt löschen
C04E: CHRGET Pointer auf nächstes Zeichen setzen
C051: X und Y Wert für die Position holen
C054: Berechnung der Position des Punktes

C089: Basis der Bit Map ist 8192 ($2000)
C094: Standard Bit Map Mode aus
C09C: Position des Bildschirms auf den Standardwert setzen
C0A1: Bildschirm löschen
C0A6: Anzahl der gedrückten Tasten ist 0

Nun können wir das Maschinenprogramm in Blöcken zusammenfassen und durch einzelne SYS Befehle ansprechen:

49152 ($C000): Einschalten des hochauflösenden-Grafikbildschirms
49169 ($C011): Löschen des hochauflösenden Grafikbildschirms
49192 ($C028): Setzen der Farbinformation
49212 ($C03C): Punkt setzen
49220 ($C044): Punkt löschen
49295 ($C08F): Standard Bit Map Mode aus

Ist das Maschinenprogramm eingelesen, so können wir damit arbeiten. Ein Beispiel: Punkt setzen an Stelle X = 50 : Y = 70

Befehl für das Maschinenprogramm:

```
X = 50
Y = 70
SYS 49212 , X , Y
```

Greifen wir jetzt noch einmal das Beispiel der Zykloidenfunktion auf und schreiben das vorher besprochene Programm (Programm auf Seite 57) auf unser eben behandeltes Maschinenprogramm um. An die Stelle der zeitraubenden FOR-NEXT Schleifen treten jetzt einfache SYS Befehle. Diese Aufrufe der Maschinenprogramme beschleunigen das Programm erheblich.
Das auf der nächsten Seite gezeigte Programm ähnelt in seinem Aufbau dem von Seite 57.

```
100 REM ZYKLOIDE MIT MASCHINENHILFSPROGRAMM
110 SYS49152
120 SYS49169
130 POKE49193,27:SYS49192
140 FORI=0TO2*πSTEP.0314159265
150 X=160-70*SIN(I)+30*SIN(5*I)
160 Y=100-70*COS(I)+30*COS(5*I)
170 SYS49212,X,Y
180 NEXT
190 IFPEEK(203)=64THEN190
200 SYS49295
210 END
```

Erläuterungen zum BASIC Programm:

110: Aufruf des Maschinenprogramms zum Einschalten des Standard Bit Map Modes
120: Löschen des hochauflösenden Grafikbildschirms
130: Farbe auswählen; hier: weiß auf grau (Farbwert 1 auf Farbwert 11; 1*16+11=27)
140: BASIC Schleife zur Berechnung der Zykloide
150: Mathematische Berechnung der X Koordinaten
160: Mathematische Berechnung der Y Koordinaten
170: Punkt mit den Koordinaten X und Y in den hochauflösenden Grafikbildschirm setzen
190: Warten auf Tastendruck
200: Standard Bit Map Mode beenden

Mit diesem Hilfsprogramm in Maschinensprache können Sie jetzt jede beliebige Grafik auf einfache und komfortable Weise, durch die Eingabe der Koordinaten eines Punktes, erstellen.
Nun stellt sich aber erneut eine Frage: Wie kann man Linien mit der Angabe eines Anfangs- und Endpunktes zeichnen?

An dieser Stelle sei als Lösung des Linienzeichenproblems ein Programmteil in Maschinensprache angegeben. Das hat den Vorteil, daß das Linienzeichnen mit wesentlich höherer Geschwindigkeit abläuft, als das mit einem BASIC Programm der Fall wäre. Das nachfolgend aufgeführte Programm ist als Ergänzung zum Hilfsprogramm auf den Seiten 62-64 gedacht und kann einfach anschließend ab Adresse 49321 ($C0A9) programmiert werden.
Die Eingabe, die dieses Programm benötigt, sieht wie folgt aus:

BASIC:

SYS 49321 , XA , YA , XE , YE

XA: X Koordinatenanfangspunkt der Gerade
YA: Y Anfangspunkt
XE: X Endpunkt
YE: Y Endpunkt

Beispiel: SYS 49321,50,35,150,174

Zeichnet man aber, wie im Beispiel erwähnt, eine Gerade vom Punkt mit den Koordinaten 50, 35 nach dem Punkt mit den Koordinaten 150, 174, so möchte ich hier noch auf eine Schwierigkeit hinweisen. Nämlich, wenn die Anfangs- und Endpunkte sich in allen vier Koordinatenwerten unterscheiden, muß eine Unterteilung berechnet werden z.B. auf drei Punkte in X Richtung kommt ein Punkt in Y Richtung. Das sind alles Rechenaufgaben komplizierterer Art, jedenfalls für BASIC, sodaß sich eine akzeptablere Lösung nur in Maschinensprache finden läßt.
Solange die Gerade sich nur in einer Koordinate ändert, beispielsweise eine horizontale Gerade, so ist die Berechnung relativ einfach.

MASCHINENSPRACHE:

```
C0A9  20 73 00    JSR $0073
C0AC  20 EB B7    JSR $B7EB
C0AF  A5 14       LDA $14
C0B1  8D 00 C9    STA $C900
C0B4  A5 15       LDA $15
C0B6  8D 01 C9    STA $C901
C0B9  8E 02 C9    STX $C902
C0BC  20 FD AE    JSR $AEFD
C0BF  20 EB B7    JSR $B7EB
C0C2  A5 14       LDA $14
C0C4  8D 03 C9    STA $C903
C0C7  A5 15       LDA $15
C0C9  8D 04 C9    STA $C904
C0CC  8E 05 C9    STX $C905
C0CF  A9 FF       LDA #$FF
C0D1  8D 0D C9    STA $C90D
C0D4  8D 0C C9    STA $C90C
C0D7  A9 00       LDA #$00
C0D9  8D 08 C9    STA $C908
C0DC  8D 0B C9    STA $C90B
C0DF  8D 07 C9    STA $C907
C0E2  38          SEC
C0E3  AD 05 C9    LDA $C905
C0E6  ED 02 C9    SBC $C902
C0E9  8D 06 C9    STA $C906
C0EC  B0 07       BCS $C0F5
C0EE  CE 07 C9    DEC $C907
C0F1  CE 08 C9    DEC $C908
C0F4  38          SEC
C0F5  AD 03 C9    LDA $C903
C0F8  ED 00 C9    SBC $C900
C0FB  8D 09 C9    STA $C909
C0FE  AD 04 C9    LDA $C904
C101  ED 01 C9    SBC $C901
C104  8D 0A C9    STA $C90A
```

```
C107 B0 1D       BCS $C126
C109 CE 0B C9    DEC $C90B
C10C 30 18       BMI $C126
C10E 0E 09 C9    ASL $C909
C111 2E 0A C9    ROL $C90A
C114 2E 0B C9    ROL $C90B
C117 0E 06 C9    ASL $C906
C11A 2E 07 C9    ROL $C907
C11D 2E 08 C9    ROL $C908
C120 4E 0D C9    LSR $C90D
C123 6E 0C C9    ROR $C90C
C126 AD 08 C9    LDA $C908
C129 4A          LSR
C12A 6A          ROR
C12B 4D 07 C9    EOR $C907
C12E 30 0A       BMI $C13A
C130 AD 0B C9    LDA $C90B
C133 4A          LSR
C134 6A          ROR
C135 4D 0A C9    EOR $C90A
C138 10 D4       BPL $C10E
C13A AD 00 C9    LDA $C900
C13D 8D 12 C9    STA $C912
C140 AD 01 C9    LDA $C901
C143 8D 13 C9    STA $C913
C146 AD 02 C9    LDA $C902
C149 8D 22 C9    STA $C922
C14C A9 80       LDA #$80
C14E 8D 11 C9    STA $C911
C151 8D 21 C9    STA $C921
C154 0A          ASL
C155 8D 10 C9    STA $C910
C158 8D 20 C9    STA $C920
C15B 8D 23 C9    STA $C923
C15E 20 C9 C1    JSR $C1C9
C161 AC 22 C9    LDY $C922
C164 AD 20 C9    LDA $C920
C167 18          CLC
C168 6D 06 C9    ADC $C906
C16B 8D 20 C9    STA $C920
```

```
C16E AD 21 C9     LDA $C921
C171 6D 07 C9     ADC $C907
C174 8D 21 C9     STA $C921
C177 AD 22 C9     LDA $C922
C17A 6D 08 C9     ADC $C908
C17D 8D 22 C9     STA $C922
C180 AE 12 C9     LDX $C912
C183 18           CLC
C184 AD 10 C9     LDA $C910
C187 6D 09 C9     ADC $C909
C18A 8D 10 C9     STA $C910
C18D AD 11 C9     LDA $C911
C190 6D 0A C9     ADC $C90A
C193 8D 11 C9     STA $C911
C196 AD 12 C9     LDA $C912
C199 6D 0B C9     ADC $C90B
C19C 8D 12 C9     STA $C912
C19F AD 13 C9     LDA $C913
C1A2 6D 0B C9     ADC $C90B
C1A5 8D 13 C9     STA $C913
C1A8 CC 22 C9     CPY $C922
C1AB D0 05        BNE $C1B2
C1AD EC 12 C9     CPX $C912
C1B0 F0 03        BEQ $C1B5
C1B2 20 C9 C1     JSR $C1C9
C1B5 AD 0C C9     LDA $C90C
C1B8 D0 09        BNE $C1C3
C1BA AD 0D C9     LDA $C90D
C1BD D0 01        BNE $C1C0
C1BF 60           RTS
C1C0 CE 0D C9     DEC $C90D
C1C3 CE 0C C9     DEC $C90C
C1C6 4C 61 C1     JMP $C161
C1C9 AD 12 C9     LDA $C912
C1CC 85 14        STA $14
C1CE AD 13 C9     LDA $C913
C1D1 85 15        STA $15
C1D3 AE 22 C9     LDX $C922
C1D6 20 54 C0     JSR $C054
C1D9 11 22        ORA ($22),Y
```

```
C1DB 91 22       STA ($22),Y
C1DD 60          RTS
C1DE AD 12 C9    LDA $C912
C1E1 85 14       STA $14
C1E3 AD 13 C9    LDA $C913
C1E6 85 15       STA $15
C1E8 AE 22 C9    LDX $C922
C1EB 20 54 C0    JSR $C054
C1EE 49 FF       EOR #$FF
C1F0 31 22       AND ($22),Y
C1F2 91 22       STA ($22),Y
C1F4 60          RTS
C1F5 A9 DE       LDA #$DE
C1F7 8D 5F C1    STA $C15F
C1FA 8D B3 C1    STA $C1B3
C1FD 20 A9 C0    JSR $C0A9
C200 A9 C9       LDA #$C9
C202 8D 5F C1    STA $C15F
C205 8D B3 C1    STA $C1B3
C208 60          RTS
```

Die beiden neu hinzugekommenen Linienfunktionen, nämlich Gerade setzen und Gerade löschen, können wie folgt angesprochen werden:

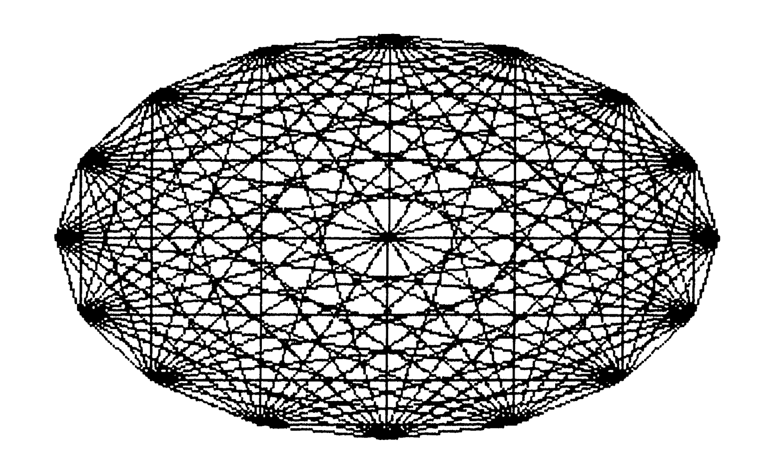
49321 ($COA9): Linie setzen
49653 ($C1F5): Linie löschen

Leider läßt sich in Maschinensprache dieses Programm nicht recht viel kürzer gestalten. Der Geschwindigkeitsunterschied gegenüber BASIC ist aber so enorm, daß man das längere Programm in Kauf nehmen kann.
Jetzt erstellen wir zur Demonstration des neuen Programmteils einmal ein Programm für ein Muster im hochauflösenden Grafikbildschirm, das sowohl in BASIC entsprechende Berechnungen erstellt, als auch auf das Maschinenprogramm zum Linienzeichnen zurückgreift.

BASIC:

```
100 REM MUSTER
110 SYS49169
120 POKE49193,114:SYS49192
130 SYS49152
140 DIMX(40),Y(40):A=0
150 FORI=0TO2*πSTEP.3926990824
160 A=A+1
170 X(A)=INT(160-150*COS(I))
180 Y(A)=INT(100-90*SIN(I))
190 NEXT
200 X(A+1)=X(1):Y(A+1)=Y(1)
210 FORI=1TOA
220 FORJ=I+1TOA
230 SYS49321,X(I),Y(I),X(J),Y(J)
240 NEXT:NEXT
250 IFPEEK(203)=64THEN250
260 SYS49295
270 END
```

Das Programm erzeugt untenstehendes Muster in der hochauflösenden Grafik:



Erläuterungen zum BASIC Programm auf Seite 72:

110: Bit Map löschen
120: Farbwerte setzen; hier: 7*16+2=114 gelb auf rot (Farbwert 7 auf Farbwert 2)
130: Standard Bit Map Mode ein
140: Dimensionierung für die Punktberechnung
150: Schleifenbeginn; der STEP beträgt 3.14159265 (PI)/8=0.392699...; der Wert muß ein ganzzahliger Teiler von PI sein damit die Eckenzahl aufgeht. Wir haben 16 Ecken damit müssen wir den STEP PI/8 wählen. Wollten wir 14 Ecken so muß der STEP PI/7=0.448798... betragen.
160: Zähler für die dimensionierten Variabeln die die Positionen der 16 Eckpunkte enthalten
170: die X Koordinatenwerte der 16 Ecken der Ellipse werden eingelesen
180: die Y Koordinatenwerte der 16 Ecken der Ellipse werden eingelesen
190: Schleifenende
200: Letzter Punkt, demnach die 17. Ecke entspricht wieder der ersten Ecke
210: Schleife für die Linienzüge
220: Schleife für die Linienzüge, denn jeder Eckpunkt soll ja mit jedem anderen Eckpunkt geradlinig verbunden werden
230: Aufrufen unseres Linienzeichenprogramms in Maschinensprache mit gleichzeitigem Einlesen der Anfangs- und Endpunktkoordinatenwerte
240: Ende der beiden Schleifen
250: Warten auf Tastendruck
260: Standard Bit Map aus

Das gezeigte BASIC Programm berechnet 16 Eckpunkte einer Ellipse und verbindet jeden Eckpunkt geradlinig mit jedem anderen Eckpunkt der Ellipse. Bereits gezeichnete Geraden werden dabei nicht nochmals gezeichnet. Die Linien werden durch das Maschinenprogramm zum Linienzeichnen berechnet und gezeichnet. Das Programm setzt die Linien mit sehr großer Geschwindigkeit unter Angabe der Anfangs- und Endkoordinaten.
Die Koordinaten dürfen in folgenden Bereichen liegen:

X-Koordinate von 0...319
Y-Koordinate von 0...199

Zur Prüfung des Linienlöschprogramms können wir im BASIC Programm auf Seite 72 die Zeile 235 einfügen. Dann nämlich werden die eben gesetzten Linien sofort wieder gelöscht:

```
235 SYS 49653,X (I),Y (I),X (J),Y (J)
```

Als Abschluß dieses Kapitels sei noch ein BASIC Programm angegeben, daß das nun häufig verwendete Maschinenprogramm aus DATA Zeilen in den Speicher überträgt.
Es läßt sich leider nicht vermeiden, daß das Programm so lang ist, dafür ist das Maschinenprogramm jedoch sehr schnell. Dazu sei nochmals erwähnt, daß BASIC, als höhere Programmiersprache, eben doch Vorteile bietet. Der Programmierkomfort wächst im gleichen Maße wie die Verständlichkeit der Programme. Ganz anders in Maschinensprache. Diese Programme sind -wenn sie nicht sehr umfangreich kommentiert sind- selbst fer den Ersteller oft unverständlich, wenn er nach einiger Zeit versucht, sich in das Programm wieder hineinzudenken.

BASIC:

```
100 REM GRAFIKHILFSPROGRAMM
110 FORI=49152TO49672
120 READA
130 POKEI,A
140 NEXT
1000 DATA   173, 24,208,  9,  8
1010 DATA   141, 24,208,173, 17
1020 DATA   208,  9, 32,141, 17
1030 DATA   208, 96,169, 32,133
1040 DATA    37,169,  0,133, 36
1050 DATA   162, 32,160,  0,145
1060 DATA    36,136,208,251,230
1070 DATA    37,202,208,244, 96
1080 DATA   169, 27,162,  0,157
1090 DATA     0,  4,157,  0,  5
1100 DATA   157,  0,  6,157,  0
1110 DATA     7,232,208,241, 96
1120 DATA    32, 78,192, 17, 34
1130 DATA   145, 34, 96, 32, 78
1140 DATA   192, 73,255, 49, 34
1150 DATA   145, 34, 96, 32,115
1160 DATA     0, 32,235,183,134
1170 DATA    35,165, 20, 41,  7
1180 DATA   170, 69, 20,133, 20
1190 DATA   169,  0,133, 34, 56
1200 DATA   106,202, 16,252,170
1210 DATA   165, 35, 41,  7,168
1220 DATA    69, 35,133, 35, 74
1230 DATA    74,101, 35, 74, 74
1240 DATA   102, 34, 74,102, 34
1250 DATA   133, 35,165, 20,101
1260 DATA    34,133, 34,165, 21
1270 DATA   101, 35,  9, 32,133
1280 DATA    35,138, 96,173, 17
1290 DATA   208, 41,223,141, 17
1300 DATA   208,173, 24,208, 41
```

```
1310 DATA   247,141, 24,208,169
1320 DATA   147, 32,210,255,169
1330 DATA     0,133,198, 96, 32
1340 DATA   115,  0, 32,235,183
1350 DATA   165, 20,141,  0,201
1360 DATA   165, 21,141,  1,201
1370 DATA   142,  2,201, 32,253
1380 DATA   174, 32,235,183,165
1390 DATA    20,141,  3,201,165
1400 DATA    21,141,  4,201,142
1410 DATA     5,201,169,255,141
1420 DATA    13,201,141, 12,201
1430 DATA   169,  0,141,  8,201
1440 DATA   141, 11,201,141,  7
1450 DATA   201, 56,173,  5,201
1460 DATA   237,  2,201,141,  6
1470 DATA   201,176,  7,206,  7
1480 DATA   201,206,  8,201, 56
1490 DATA   173,  3,201,237,  0
1500 DATA   201,141,  9,201,173
1510 DATA     4,201,237,  1,201
1520 DATA   141, 10,201,176, 29
1530 DATA   206, 11,201, 48, 24
1540 DATA    14,  9,201, 46, 10
1550 DATA   201, 46, 11,201, 14
1560 DATA     6,201, 46,  7,201
1570 DATA    46,  8,201, 78, 13
1580 DATA   201,110, 12,201,173
1590 DATA     8,201, 74,106, 77
1600 DATA     7,201, 48, 10,173
1610 DATA    11,201, 74,106, 77
1620 DATA    10,201, 16,212,173
1630 DATA     0,201,141, 18,201
1640 DATA   173,  1,201,141, 19
1650 DATA   201,173,  2,201,141
1660 DATA    34,201,169,128,141
1670 DATA    17,201,141, 33,201
1680 DATA    10,141, 16,201,141
1690 DATA    32,201,141, 35,201
```

```
1700 DATA    32,201,193,172, 34
1710 DATA   201,173, 32,201, 24
1720 DATA   109,  6,201,141, 32
1730 DATA   201,173, 33,201,109
1740 DATA     7,201,141, 33,201
1750 DATA   173, 34,201,109,  8
1760 DATA   201,141, 34,201,174
1770 DATA    18,201, 24,173, 16
1780 DATA   201,109,  9,201,141
1790 DATA    16,201,173, 17,201
1800 DATA   109, 10,201,141, 17
1810 DATA   201,173, 18,201,109
1820 DATA    11,201,141, 18,201
1830 DATA   173, 19,201,109, 11
1840 DATA   201,141, 19,201,204
1850 DATA    34,201,208,  5,236
1860 DATA    18,201,240,  3, 32
1870 DATA   201,193,173, 12,201
1880 DATA   208,  9,173, 13,201
1890 DATA   208,  1, 96,206, 13
1900 DATA   201,206, 12,201, 76
1910 DATA    97,193,173, 18,201
1920 DATA   133, 20,173, 19,201
1930 DATA   133, 21,174, 34,201
1940 DATA    32, 84,192, 17, 34
1950 DATA   145, 34, 96,173, 18
1960 DATA   201,133, 20,173, 19
1970 DATA   201,133, 21,174, 34
1980 DATA   201, 32, 84,192, 73
1990 DATA   255, 49, 34,145, 34
2000 DATA    96,169,222,141, 95
2010 DATA   193,141,179,193, 32
2020 DATA   169,192,169,201,141
2030 DATA    95,193,141,179,193
2040 DATA    96
```

Erläuterungen zum BASIC Programm:

110: Schleife zum Einlesen der Daten von
     Adresse 49152 ($C000) bis Adresse 49672
     ($C208)
120: Werte aus den DATA Zeilen auslesen
130: Speicherstellen beschreiben
140: Ende der Schleife

Das Maschinenprogramm, das mit dem eben aufgezeigten BASIC Ladeprogramm eingelesen wird, bietet folgendes:

Grafik Hilfsprogramm Zusammenfassung:

- Einschalten der hochauflösenden Grafik und
  Bit Map ab Adresse 8192 ($2000) positionieren
  Befehl: SYS 49152
- Löschen des hochauflösenden Grafikbildschirms
  Befehl: SYS 49169
- Setzen der Farbinformation
  Anwahl der Farben: POKE 49193,VG*16 + HG
  VG: Farbwert der gesetzten Punkte (0...15)
  HG: Farbwert der gelöschten Punkte (0...15)
  Befehl: SYS 49192
- Punkt setzen
  X darf im Bereich von 0...319 sein
  Y darf im Bereich von 0...199 sein
  Befehl: SYS 49212,X,Y
- Punkt löschen
  X = 0...319, Y = 0...199
  Befehl: SYS 49220,X,Y
- Linie zeichnen
  XA,YA Koordinaten des Anfangspunktes
  XE,YE Koordinaten des Endpunktes
  Befehl: SYS 49321,XA,YA,XE,YE
- Linie löschen
  Befehl: SYS 49653,XA,YA,XE,YE
- Ausschalten der hochauflösenden Grafik

4.2 Multi Color Bit Map Mode

Der Multi Color Bit Map Mode entspricht fast dem Standard Bit Map Mode, jedoch können die im hochauflösenden Grafikbildschirm gesetzten Punkte verschiedene Farben annehmen. Das bedeutet, daß wir zum Beispiel auf grauem Hintergrund in der oberen Hälfte des Grafikbildschirms einen roten Kreis und in der unteren Hälfte einen gelben Kreis zeichnen können. Die Hintergrundfarbe läßt sich hier wie im Standard Character Mode durch das Hintergrundfarbregister 0 kontrollieren.
Im Multi Color Bit Map Mode können wir für jede 8 x 8 Matrix drei verschiedene Farben der Punkte anwählen.
Ähnlich wie im Multi Color Character Mode, in dem man verschiedenfarbige Zeichen programmieren kann, ist auch hier die Auflösung in X-Richtung halbiert. Das hat seinen Grund in der Auswahl der Farben für einen Punkt. Zwei Bits bestimmen also woher die Farbe eines Punktes genommen wird. In X-Richtung stehen uns somit nur 159 Punkte (Doppelpunkte) zur Verfügung. In Y-Richtung besitzen wir weiterhin 199 Zeilen.
Die Doppelpunktdarstellung läßt es auch mit einem normalen Farbfernsehgerät zu, die Farbe eines gesetzten Punktes eindeutig zu erkennen. Wäre ein einzelner Punkt rot und der Punkt daneben gelb, so würden wahrscheinlich auf dem normalen Farbfernseher beide gleich bunt aussehen. Ein guter Farbmonitor hingegen könnte die beiden Punkte in ihrer wirklich definierten Farbe darstellen.
Kontrolliert wird der Multi Color Bit Map Mode durch Bit 5 der Adresse 53265 ($D011) und durch Bit 4 der Adresse 53270 ($D016). Sind beide Bits auf 1 gesetzt, so ist der Multi Color Bit Map Mode eingeschaltet. Sind beide auf 0 so ist der Multi Color Bit Map Mode ausgeschaltet. Ähnlich wie beim Standard Character Mode muß auch hier für die

Bit Map (Speicherbereich in dem steht, welche Punkte gesetzt und welche nicht gesetzt sind) ein Speicherbereich von 8000 Byte zur Verfügung gestellt werden. Für das folgende Beispiel wählen wir am besten wieder den Bereich von 8192-16191 ($2000-$3F3F) aus.

BASIC:

```
POKE 53265,PEEK (53265) OR 32
POKE 53270,PEEK (53270) OR 16
; schaltet den Multi Color Bit Map Mode ein
```

MASCHINENSPRACHE:

```
    MCBMM1 =$D011
    MCBMM2 =$D016
LDA MCBMM1  ; Wert laden
ORA #$20    ; Bit 5 setzen
STA MCBMM1  ; abspeichern
LDA MCBMM2  ; Wert laden
ORA #$10    ; Bit 4 setzen
STA MCBMM1  ; Multi Color Bit Map Mode ein
RTS
```

BASIC:

```
POKE 53265,PEEK (53265) AND 223
POKE 53270,PEEK (53270) AND 239
; schaltet den Multi Color Bit Map Mode aus
```

MASCHINENSPRACHE:

```
    MCBMM1 =$D011
    MCBMM2 =$D016
LDA MCBMM1  ; Wert laden
```

```
AND #$DF    ; Bit 5 löschen
STA MCBMM1 ; abspeichern
LDA MCBMM2 ; Wert laden
AND #$EF    ; Bit 4 löschen
STA MCBMM2 ; Multi Color Bit Map Mode aus
RTS
```

BASIC:

```
POKE 53272,PEEK (53272) OR 8
; setzt die Position der Bit Map ab Adresse
  8192 ($2000)
```

MASCHINENSPRACHE:

```
    POS =$D018
LDA POS      ; Positionswert laden
ORA #$08     ; Bit 3 setzen
STA POS      ; setzt die Bit Map ab 8192 ($2000)
RTS
```

Die beiden Bits eines Punktes bestimmen woher die Farbinformation, d.h. die Farbe des Punktes, genommen wird:

BITS	FARBINFORMATION AUS	ADRESSE
00	Hintergrundfarbregister 0	53281 ($D021)
01	Bit 4...7 Video-RAM	ab 1024 ($0400)
10	Bit 0...3 Video-RAM	ab 1024 ($0400)
11	Bit 0...3 Farb-RAM	ab 55296 ($D800)

Auf der nächsten Seite ist dazu ein einfaches Beispielprogramm aufgeführt:

BASIC:

```
100 REM MULTI COLOR BIT MAP MODE
110 POKE53272,PEEK(53272)OR8
120 POKE53265,PEEK(53265)OR32
130 POKE53270,PEEK(53270)OR16
140 FORI=8192TO16191
150 POKEI,0
160 NEXT
170 POKE53281,11
180 POKE1025,1*16:POKE1026,4
190 POKE55299,3
200 POKE8192,0:POKE8193,0
210 POKE8200,85:POKE8201,85
220 POKE8208,170:POKE8209,170
230 POKE8216,255:POKE8217,255
240 IFPEEK(203)=64THEN240
250 POKE53265,PEEK(53265)AND223
260 POKE53270,PEEK(53270)AND239
270 POKE53272,PEEK(53272)AND247
280 PRINT"♥";:POKE198,0
290 END
```

Das Programm setzt in die linke obere Ecke Punkte in verschiedenen Farben. Ganz links ist dieselbe Farbe wie die Hintergrundfarbe (Farbwert 11), die Punkte sind also nicht gesetzt. Rechts daneben sind zwei Zeilen des Grafikbildschirms weiß (Farbwert 1). Wiederum daneben zwei purpurfarbige Zeilen. Schließlich zwei Zeilen türkis (CYN Farbwert 3).

Erläuterungen zum BASIC Programm:

110: Position der Bit Map auf 8192 ($2000) setzen
120: Multi Color Bit Map Mode ein

130: Multi Color Bit Map Mode ein
140: Schleife zum Löschen des Grafikbildschirms
170: Farbe grau (Farbwert 11) in Farbregister 0
180: Farbwert 1 (weiß) in Bit 4...7 also
0001 XXXX der entsprechenden 8 x 8 Gruppe
in der sich die gesetzten Punkte befinden
dasselbe nur für Bit 0...3; hier: Farbwert
4 (purpur)
190: Farbe türkis in Bit 0...3 des Farb-RAM
200: die ersten zwei Zeilen der Bit Map also
links oben mit der Hintergrundfarbe be-
setzen
210: Bitkonfiguration 0101 0101 = 85 ($55)
das bedeutet die Farbe dieser Punkte
kommt von Bit 4...7 des Video-RAM's, nämlich
Adresse 1025 ($0401)
220: Bitmuster 1010 1010 = 170 ($AA); Farbe dieser
Punkte kommt aus Bit 0...3 der entsprechen-
den 8 x 8 Gruppe im Video-RAM, hier: 1026
($0402)
230: Bitmuster 1111 1111 = 255 ($FF); Farbe der
Punkte kommt aus Bit 0...3 des Farb-RAM's;
auch hier wieder für jede 8 x 8 Matrix
bei uns 55299 ($D803)
240: Wartet auf gedrückte Taste
250: Multi Color Bit Map Mode aus
260: Multi Color Bit Map Mode aus
270: Position des Bildschirms wieder auf den
Standardwert setzen
280: Bildschirm löschen und Anzahl der gedrück-
ten Tasten auf 0 zurücksetzen

Auch hier stellt sich die Frage, wie man auf komfortable Weise die Punkte unter Angabe der Koordinaten und unter Angabe, woher die Farbe kommen soll, eingeben kann. Dazu können wir unser BASIC Programm auf Seite 56 benutzen. Allerdings ist eine Ergänzung nötig. In X-Richtung müssen wir nun in Abhängigkeit der Farbe, nur das eine oder nur das andere Bit oder beide Bits setzen, denn

wir übergeben mit diesen beiden Bits auch die Information woher die Farbe kommen soll. Aus diesem Grund wählen wir im nachfolgenden BASIC Programm durch die Zahlen 1, 2, 3 woher die Farbe genommen wird. Der Einfachheit halber benutzen wir für den ganzen Bildschirm dieselben vier Farben der Punkte. Theoretisch ist es aber möglich, für jede 8 x 8 Matrix vier eigene, unterschiedliche Farben zu definieren.

BASIC:

```
100 REM MULTI COLOR BIT MAP MODE
110 POKE53272,PEEK(53272)OR8
120 POKE53265,PEEK(53265)OR32
130 POKE53270,PEEK(53270)OR16
140 FORI=8192TO16191
150 POKEI,0
160 NEXT
170 FORI=0TO999
180 POKEI+1024,1*16+2
185 POKEI+55296,7
190 NEXT
200 FORX=0TO159
210 Y=10:F=1
220 GOSUB2000
230 NEXT
240 FORX=0TO159
250 Y=15:F=2
260 GOSUB2000
270 NEXT
280 FORX=0TO159
290 Y=20:F=3
300 GOSUB2000
310 NEXT
900 IFPEEK(203)=64THEN900
910 POKE53265,PEEK(53265)AND223
920 POKE53270,PEEK(53270)AND239
930 POKE53272,PEEK(53272)AND247
```

```
940 PRINT"[CLR]";:POKE198,0
950 END
1000 RE=INT(Y/8)
1010 SP=INT(X1/8)
1020 ZE=YAND7
1030 BI=7-(X1AND7)
1040 BY=8192+RE*320+SP*8+ZE
1050 POKEBY,PEEK(BY)OR2↑BI
1060 RETURN
2000 REM FARBUNTERSCHEIDUNG
2010 X1=X*2
2020 IFF=1THENX1=X1+1:GOSUB1000:RETURN
2030 IFF=2THENGOSUB1000:RETURN
2040 IFF=3THENGOSUB1000:X1=X1+1:GOSUB1000:RETURN
```

Das vorliegende BASIC Programm zeichnet im Multi Color Bit Map Mode drei horizontale Linien in drei verschiedenen Farben.

Erläuterungen zum BASIC Programm:

110: Position der Bit Map auf 8192 ($2000) setzen
120: Multi Color Bit Map Mode ein
130: Multi Color Bit Map Mode ein
140: Schleife zum Löschen des hochauflösenden Grafikbildschirms
170: Setzen der Farben für den gesamten Bildschirm
200: Schleife für die erste Gerade, diese Gerade hat die Farbe weiß (Farbwert 1); mit der Variabeln F (F = 1...3) kann man anwählen, woher die Punkte ihre Farbe nehmen; bei F=1 bestimmen Bit 4...7 des Video-RAM's, bei F=2 Bit 0...3 des Video-RAM's und bei F=3 Bit 0...3 des Farb-RAM's die Farbe
220: Sprung ins Farbauswahlunterprogramm und

von dort in das Punktsetzprogramm
240: Zweite Gerade, Farbe rot (Farbwert 2)
280: Dritte Gerade, Farbe gelb (Farbwert 7)
900: Warten auf Tastendruck
910: Multi Color Bit Map Mode aus
920: Multi Color Bit Map Mode aus
930: Position des Bildschirms auf Standardwert setzen
940: Bildschirm löschen, Anzahl der gedrückten Tasten auf 0 setzen
1000: Punktsetzroutine; Reihenberechnung
1010: Spaltenberechnung
1020: Zeilenberechnung
1030: Bitberechnung
1040: Byteberechnung
1050: Einspeichern der berechneten Daten und Setzen der entsprechenden Punkte
1060: Rücksprung aus dem Unterprogramm
2000: Farbauswahlunterprogramm
2010: Der X Wert wird hier mit zwei multipliziert, da wir in X-Richtung 159 Doppelpunkte besitzen, die aber, wegen der Farben einzeln gesetzt werden müssen.
2020: Ist F=1 dann wird die X-Einzelpunktkoordinate um eins erhöht und das zweite Bit gesetzt (Bitmuster: 01)
2030: Bei F=2 wird nur das erste Bit gesetzt
2040: Ist F=3 so wird das erste und das zweite Bit gesetzt, also ein doppelter Sprung in das Punktsetz-Unterprogramm

Nun können wir zum Löschen des Grafikbildschirms und zum Einspeichern der Farbinformation Maschinenprogramme benutzen, was die Arbeitsgeschwindigkeit erheblich erhöht. Besprechen wir im folgenden also diverse kleinere Maschinenprogramme gewissermaßen als Unterstützung zum BASIC Programm.

BASIC:

```
POKE 53265,PEEK (53265) OR 32
POKE 53270,PEEK (53270) OR 16
; Multi Color Bit Map Mode ein
POKE 53272,PEEK (53272) OR 8
; Position der Bit Map auf 8192 ($2000) setzen
```

MASCHINENSPRACHE:

```
    MCBMM1 =$D011
    MCBMM2 =$D016
    POS    =$D018
LDA MCBMM1  ; Wert laden
ORA #$20    ; Bit 5 setzen
STA MCBMM1  ; abspeichern
LDA MCBMM2  ; Wert laden
ORA #$10    ; Bit 4 setzen
STA MCBMM2  ; Multi Color Bit Map Mode ein
LDA POS     ; Positionswert laden
ORA #$08    ; Bit 3 setzen
STA POS     ; Bit Map Position ist 8192 ($2000)
RTS
```

BASIC:

```
FOR I=8191 TO 16191
POKE I,0
NEXT
; löschen  des hochauflösenden  Grafikbildschirms
```

MASCHINENSPRACHE:

```
LDA #$20    ; H-Byte der Adr. 8192 ($2000)
STA $25     ; in freie Adresse abspeichen
LDA #$00    ; L-Byte laden
```

```
      STA $24       ; abspeichern
      LDX #$20      ; X-Schleifenzähler
XLOOP LDY #$00      ; Y-Schleifenzähler
YLOOP STA ($24),Y   ; 0 in Bit Map einspeichern
      INY           ; Y=Y+1
      BNE YLOOP     ; Y<>0 dann nach YLOOP springen
      INC $25       ; H-Byte +1
      DEX           ; X=X-1
      BNE XLOOP     ; X<>0 dann nach XLOOP springen
      RTS
```

BASIC:

```
FOR I=0 TO 999
POKE I+1024,1*16+2
POKE I+55296,7
NEXT
; Schleife zum Einlesen der Farbinformation
```

MASCHINENSPRACHE:

```
      LDA #$17      ; Werte für  weiß  und gelb
      LDX #$00      ; X-Schleifenzähler
LOOP1 STA $0400,X   ; das gesamte Video-RAM mit
      STA $0500,X   ; dem Wert für  weiß  (Farbwert
      STA $0600,X   ; 1) und gelb (Farbwert 7)
      STA $0700,X   ; beschreiben
      INX           ; X=X+1
      BNE LOOP1     ; X<>0 dann nach LOOP1 springen
      LDA #$08      ; Farbe orange (Farbwert 8)
LOOP2 STA $D800,X   ; das gesamte Farb-RAM mit dem
      STA $D900,X   ; Wert für orange beschreiben;
      STA $DA00,X   ; Bit 0...3 entsprechen der
      STA $DB00,X   ; Farbe
      INX           ; X=X+1
      BNE LOOP2     ; X<>0 dann nach LOOP2 springen
      RTS
```

BASIC:

```
POKE 53270,PEEK (53270) AND 223
; Multi Color Bit Map Mode aus
POKE 53272,PEEK (53272) AND 247
; Bildschirm auf Standardwert zurücksetzen
PRINT 'CLR/HOME';:POKE 198,0
; Bildschirm löschen;  Anzahl der gedrückten
  Tasten auf 0 setzen
```

MASCHINENSPRACHE:

```
    MCBMM1 =$D011
    MCBMM2 =$D016
    POS    =$D018
    PRINT  =$FFD2
    ANZTAS =$00C6
LDA MCBMM1  ; Wert laden
AND #$DF    ; Bit 5 löschen
STA MCBMM1  ; abspeichern
LDA MCBMM2  ; Wert laden
AND #$EF    ; Bit 4 löschen
STA MCBMM2  ; Multi Color Bit Map Mode aus
LDA POS     ; Positionswert laden
AND #$F7    ; Bit 3 löschen
STA POS     ; Bildschirm auf Standardwert setzen
LDA #$93    ; Wert für  CLR/HOME laden
JSR PRINT   ; Bildschirm löschen
LDA #$00    ; Wert 0 laden
STA ANZTAS  ; Anzahl der gedrückten  Tasten auf 0
RTS
```

BASIC:

```
RE = INT (Y/8)                   ; Reihenberechnung
SP = INT (X1/8)                  ; Spaltenberechnung
ZE = Y AND 7                     ; Zeilenberechnung
```

```
BI = 7-(X1 AND 7)                  ; Bitberechnung
BY = 8192 + RE*320 + SP*8 + ZE ; Byteberechnung
POKE BY,PEEK (BY) OR 2 ↑ BI     ; Punkt setzen
X1 = X*2                           ; X Koordinate *2
IF F=1 THEN X1=X1+1...
IF F=2 THEN... ... ...
; Farbunterscheidung
```

Das Programm zur Farbunterscheidung in Maschinensprache ist ein etwas länger als die übrigen. Je nachdem welche Farbe Sie auswählen, muß das Programm entweder das Bitmuster 01 oder 10 oder 11 in die Bit Map schreiben:

MASCHINENSPRACHE:

```
            CHRGET =$0073
            KOLES  =$B7EB
POSET LDA $14      ; L-Byte der X Koordinate
      AND #$07     ; Bit 4...7 löschen
      TAX          ; Akku (A) nach X
      EOR $14      ; A EOR Adresse $14
      STA $14      ; abspeichern
      LDA #$00     ; Wert 0 laden
      STA $22      ; in Adresse $22 abspeichern
      SEC          ; Carry Bit setzen (C=1)
LOOP  ROR          ; A ein Bit nach rechts
      DEX          ; X=X-1
      BPL LOOP     ; X=255? nein, dann nach LOOP
      TAX          ; Akku nach X
      LDA $23      ; Y-Koordinate laden
      AND #$07     ; Bit 4...7 löschen
      TAY          ; Akku nach Y
      EOR $23      ; A EOR Adresse $23
      STA $23      ; in Adresse $23 abspeichern
      LSR          ; A um zwei Bit nach rechts
      LSR          ; entspricht geteilt durch 4
      ADC $23      ; Adr. $23 addieren
```

```
      LSR           ; A zwei Bit nach rechts ist
      LSR           ; geteilt durch 4
      ROR $22       ; Adr. $22 ein Bit nach rechts
      LSR           ; A ein Bit nach rechts
      ROR $22       ; Adr. $22 ein Bit nach rechts
      STA $23       ; in Adr. $23 abspeichern
      LDA $14       ; Adr. $14 laden
      ADC $22       ; Adr. $22 addieren
      STA $22       ; abspeichern
      LDA $15       ; H-Byte der X Koordinate laden
      ADC $23       ; Adr. $23 addieren
      ORA #$20      ; Bit 5 setzen
      STA $23       ; abspeichern
      TXA           ; X nach Akku
      RTS           ; Rücksprung
FRBUN JSR CHRGET    ; CHRET holt nächstes  Zeichen
      JSR KOLES     ; Holt X und Y Koordinaten
      STX $23       ; Y Koordinate abspeichern
      LDY #$01      ; Farbauswahl F = 1...3
      CLC           ; Carry Bit löschen  (C=0)
      ASL $14       ; L-Byte der X Koordinate mal 2
      BCC NROL      ; >255?, nein, dann nach NROL
      ROL $15       ; H-Byte der X Koordinate
NROL  CPY #$01      ; F mit 1 vergleichen
      BNE FN1       ; F<>1 dann Sprung nach FN1
      INC $14       ; L-Byte $14 +1
      LDA $14       ; mit Adr. $14 laden
      BNE NINC      ; A<> 0 dann nach NINC springen
      INC $15       ; H-Byte +1
NINC  JSR SET       ; Punkt berechnen und setzen
      RTS           ; Rücksprung
FN1   CPY #$02      ; F=2?
      BNE FN2       ; F<> 2 dann Sprung nach FN2
      JSR SET       ; Punkt berechnen und setzen
      RTS           ; Rücksprung
FN2   CPY #$03      ; F=3?
      BNE FN3       ; F<> 3 dann zurück  nach BASIC
      LDA $15       ; H-Byte der X Koordinate
      STA $C900     ; abspeichern
      LDA $14       ; L-Byte der X Koordinate
```

```
        STA $C901    ; abspeichern
        LDA $23      ; Y Koordinate laden
        STA $C902    ; abspeichern
        JSR SET      ; ersten Punkt setzen
        LDA $C901    ; L-Byte der X Koordinate
        STA $14      ; abspeichern
        LDA $C900    ; H-Byte der X Koordinate
        STA $15      ; abspeichern
        LDA $C902    ; Y Koordinate laden
        STA $23      ; abspeichern
        INC $14      ; zweites Bit auch errechnen
        BNE NHIGH    ; <>255 dann nach NHIGH
        INC $15      ; H-Byte +1
        JSR SET      ; Punkt berechnen und setzen
        RTS          ; Rücksprung
SET     JSR POSET    ; Punkt berechnen
        ORA ($22),Y  ; entsprechende Bits setzen
        STA ($22),Y  ; Punkte setzen
FN3     RTS          ; Rücksprung
```

Das oben aufgeführte Maschinenprogramm unterscheidet zwischen drei Farben. Beim Aufruf muß man also drei Informationen geben:

- X Koordinate des Punktes (X = 0...159)
- Y Koordinate des Punktes (Y = 0...199)
- F woher die Farbe des Punktes kommt (F = 1...3)

Ist F=1, so kommt die Farbinformation aus den oberen vier Bits der zutreffenden 8 x 8 Gruppe im Video-RAM. Ist F=2, so kommt die Information aus den unteren vier Bits des Video-RAM's. Bei F=3 wird die Farbe aus den unteren vier Bits des Farb-RAM's geholt.
Auf der nächsten Seite ist das komplette Maschinenprogramm abgedruckt, daß wir in Einzelschritten besprochen und mit BASIC verglichen haben. Wir wählen als Startadresse für dieses Hilfsprogramm 49152 ($C000). Das Programm verfügt auch über schnelle Routinen wie zum Beispiel zum

Bit Map löschen und zum Beschreiben mit der Farbinformation. Eine Bemerkung noch zur Farbe: in unserem Hilsfprogramm in Maschinensprache benutzen wir für den gesamten Bildschirm dieselben drei Punktfarben. Mit etwas größerem Aufwand kann man ebenfalls durch ein Maschinenprogramm für jede 8 x 8 Matrix vier eigene unterschiedliche Farben definieren.

MASCHINENSPRACHE:

```
C000 AD 11 D0    LDA $D011
C003 09 20       ORA #$20
C005 8D 11 D0    STA $D011
C008 AD 16 D0    LDA $D016
C00B 09 10       ORA #$10
C00D 8D 16 D0    STA $D016
C010 AD 18 D0    LDA $D018
C013 09 08       ORA #$08
C015 8D 18 D0    STA $D018
C018 60          RTS
C019 A9 20       LDA #$20
C01B 85 25       STA $25
C01D A9 00       LDA #$00
C01F 85 24       STA $24
C021 A2 20       LDX #$20
C023 A0 00       LDY #$00
C025 91 24       STA ($24),Y
C027 C8          INY
C028 D0 FB       BNE $C025
C02A E6 25       INC $25
C02C CA          DEX
C02D D0 F4       BNE $C023
C02F 60          RTS
C030 A9 17       LDA #$17
C032 A2 00       LDX #$00
C034 9D 00 04    STA $0400,X
C037 9D 00 05    STA $0500,X
```

```
C03A 9D 00 06    STA $0600,X
C03D 9D 00 07    STA $0700,X
C040 E8          INX
C041 D0 F1       BNE $C034
C043 A9 08       LDA #$08
C045 A2 00       LDX #$00
C047 9D 00 D8    STA $D800,X
C04A 9D 00 D9    STA $D900,X
C04D 9D 00 DA    STA $DA00,X
C050 9D 00 DB    STA $DB00,X
C053 E8          INX
C054 D0 F1       BNE $C047
C056 60          RTS
C057 AD 11 D0    LDA $D011
C05A 29 DF       AND #$DF
C05C 8D 11 D0    STA $D011
C05F AD 16 D0    LDA $D016
C062 29 EF       AND #$EF
C064 8D 16 D0    STA $D016
C067 AD 18 D0    LDA $D018
C06A 29 F7       AND #$F7
C06C 8D 18 D0    STA $D018
C06F A9 93       LDA #$93
C071 20 D2 FF    JSR $FFD2
C074 A9 00       LDA #$00
C076 85 C6       STA $C6
C078 60          RTS
C079 A5 14       LDA $14
C07B 29 07       AND #$07
C07D AA          TAX
C07E 45 14       EOR $14
C080 85 14       STA $14
C082 A9 00       LDA #$00
C084 85 22       STA $22
C086 38          SEC
C087 6A          ROR
C088 CA          DEX
C089 10 FC       BPL $C087
C08B AA          TAX
```

```
C08C A5 23        LDA $23
C08E 29 07        AND #$07
C090 A8           TAY
C091 45 23        EOR $23
C093 85 23        STA $23
C095 4A           LSR
C096 4A           LSR
C097 65 23        ADC $23
C099 4A           LSR
C09A 4A           LSR
C09B 66 22        ROR $22
C09D 4A           LSR
C09E 66 22        ROR $22
C0A0 85 23        STA $23
C0A2 A5 14        LDA $14
C0A4 65 22        ADC $22
C0A6 85 22        STA $22
C0A8 A5 15        LDA $15
C0AA 65 23        ADC $23
C0AC 09 20        ORA #$20
C0AE 85 23        STA $23
C0B0 8A           TXA
C0B1 60           RTS
C0B2 20 73 00     JSR $0073
C0B5 20 EB B7     JSR $B7EB
C0B8 86 23        STX $23
C0BA A0 01        LDY #$01
C0BC 18           CLC
C0BD 06 14        ASL $14
C0BF 90 02        BCC $C0C3
C0C1 26 15        ROL $15
C0C3 C0 01        CPY #$01
C0C5 D0 0C        BNE $C0D3
C0C7 E6 14        INC $14
C0C9 A5 14        LDA $14
C0CB D0 02        BNE $C0CF
C0CD E6 15        INC $15
C0CF 20 0C C1     JSR $C10C
C0D2 60           RTS
```

```
C0D3 C0 02       CPY #$02
C0D5 D0 04       BNE $C0DB
C0D7 20 0C C1    JSR $C10C
C0DA 60          RTS
C0DB C0 03       CPY #$03
C0DD D0 34       BNE $C113
C0DF A5 15       LDA $15
C0E1 8D 00 C9    STA $C900
C0E4 A5 14       LDA $14
C0E6 8D 01 C9    STA $C901
C0E9 A5 23       LDA $23
C0EB 8D 02 C9    STA $C902
C0EE 20 0C C1    JSR $C10C
C0F1 AD 01 C9    LDA $C901
C0F4 85 14       STA $14
C0F6 AD 00 C9    LDA $C900
C0F9 85 15       STA $15
C0FB AD 02 C9    LDA $C902
C0FE 85 23       STA $23
C100 E6 14       INC $14
C102 A5 14       LDA $14
C104 D0 02       BNE $C108
C106 E6 15       INC $15
C108 20 0C C1    JSR $C10C
C10B 60          RTS
C10C 20 79 C0    JSR $C079
C10F 11 22       ORA ($22),Y
C111 91 22       STA ($22),Y
C113 60          RTS
```

Erläuterungen zum Maschinenprogramm:

C000: Multi Color Bit Map Mode einschalten
C010: Position der Bit Map auf Adresse 8192 ($2000) setzen
C019: Hochauflösenden Grafikbildschirm löschen

C030: Farbinformation im Video-RAM setzen
C031: Farbwert für Bit 0...7 im Video-RAM
C043: Farbinformation im Farb-RAM setzen
C044: Farbwert für Bit 0...3 im Farb-RAM
C057: Multi Color Bit Map Mode aus
C067: Position des Bildschirms auf Standardwert
C06F: Bildschirm löschen
C074: Anzahl der gedrückten Tasten auf 0
C079: Punktberechnungsroutine
C0B2: CHRGET holt nächstes Zeichen
C0B5: X und Y Koordinatenwerte einlesen
C0BB: Kontrollwert der Farbe F (F = 1...3)
C10C: Punktsetzroutine

Fassen wir nun alles zusammen und errechnen die einzelnen Adressen der SYS Befehle, damit wir unser Hilfsprogramm einsetzen können:

49152 ($C000): Einschalten des Multi Color Bit Map Mode Bildschirms
49177 ($C019): Löschen des hochauflösenden Grafikbildschirms
40200 ($C030): Setzen der Farbinformation in Video- und Farb-RAM
49239 ($C057): Ausschalten des Multi Color Bit Map Mode
49330 ($C0B2): Punkt setzen

Die Farbwerte mit denen Video-RAM und Farb-RAM beschrieben werden, können sie verändern mit:

```
POKE 49201,A*16 + B ; A ist Punktfarbe bei F=1
                    ; B ist Punktfarbe bei F=2
POKE 49220,C        ; C ist Punktfarbe bei F=3
```

Um unser Hilfsprogramm in Maschinenprogramm auch einsetzen zu können, wollen wir ein kleines Beispielprogramm besprechen. Unser Programm soll auf einem grauen Bildschirm einen weißen Kreis, eine gelbe Ellipse und eine hellgrüne Zykloide zeich-

nen. Dazu legen wir zuerst die Farbwerte fest:

```
weiß     : Farbwert 1  bei F = 1
gelb     : Farbwert 7  bei F = 2
hellgrün : Farbwert 13 bei F = 3
```

Diese Farbwerte müssen dann eingespeichert werden:

```
POKE 49201,1*16+7 ; Video-RAM Wert
POKE 49220,13     ; Farb-RAM Wert
```

Weiterhin müssen wir den Farbkontrollwert F anwählen. Ist in unserem Beispiel F=1, so wäre die Punktfarbe weiß (Farbwert 1). Bei F=2 ist die Punktfarbe gelb (Farbwert 7) und bei F=3 ist die Punktfarbe hellgrün (Farbwert 13):

```
POKE 49339,F      ; Farbkontrollwert F; F=1...3
```

BASIC:

```
100 REM MULTI COLOR BIT MAP MODE BEISPIEL
105 POKE53281,11
110 SYS49177
120 SYS49152
130 POKE49201,1*16+7
140 POKE49220,13
150 SYS49200
160 POKE49339,1
170 FORI=0TO2*πSTEP.0314159265
180 X=40-25*SIN(I)
190 Y=60-50*COS(I)
200 SYS49330,X,Y
210 NEXT
220 POKE49339,2
230 FORI=0TO2*πSTEP.0314159265
240 X=70-50*SIN(I)
250 Y=60-50*COS(I)
260 SYS49330,X,Y
```

```
270 NEXT
280 POKE49339,3
290 FORI=0TO2*πSTEP.0314159265
300 X=90-30*SIN(I)+20*SIN(5*I)
310 Y=120-30*COS(I)+20*COS(5*I)
320 SYS49330,X,Y
330 NEXT
1000 IFPEEK(203)=64THEN1000
1010 SYS49239
1020 END
```

Erläuterungen zum BASIC Programm:

105: Hintergrundfarbe setzen; hier Farbwert 11, grau
110: Hochauflösenden Grafikbildschirm löschen
120: Multi Color Bit Map Mode einschalten
130: Farbinformation für das Video-RAM einspeichern; hier Farbe 1 = weiß (Farbwert 1) Farbe 2 ist gelb (Farbwert 7)
140: Farbinformation für Farbe 3 im Farb-RAM setzen; hier hellgrün (Farbwert 13)
150: Aufrufen der Maschinenroutine für das Einspeichern der Farbinformationen in Video- und Farb-RAM
160: Farbkontrollregister F=1
170: Schleife fuer die Berechnung des Kreises
180: Mathematische Berechnung der X Koordinate
190: Berechnung der Y Koordinate
200: Punkt setzen und zwar weiß, da F=1 ist
210: Schleifenende
220: Farbkontrollregister F=2
230: Schleife für die Berechnung der Ellipse
240: Berechnung der X Koordinate
250: Berechnung der Y Koordinate
260: Punkt setzen und zwar gelb, da F=2
270: Ende der Schleife

280: Farbkontrollregister F=3
290: Schleife zur Berechnung der Zykloide
300: X Koordinatenberechnung
310: Y Koordinatenberechnung
320: Punkt setzen; hier hellgrün, da F=3
330: Ende der Schleife
1000: Wartet auf Tastendruck
1010: Multi Color Bit Map Mode aus

Es sei hier nochmal verdeutlicht, daß das Farbkontrollregister F, bei unserem Maschinenprogramm Adresse 49330 ($C0BB), nur die Aufgabe besitzt, zu bestimmen woher die Farbe kommt. Das Kontrollregister kann drei Werte annehmen, deren Auswirkung in der anschließenden Tabelle aufgeführt sind:

F	Farbe kommt aus	Adresse
1	Bit 4...7 Video-RAM	ab 1024 ($0400)
2	Bit 0...3 Video-RAM	ab 1024 ($0400)
3	Bit 0...3 Farb-RAM	ab 55296 ($D800)

Zum Ändern der Farben können Sie zum Beispiel Zeile 130 des oben angeführten BASIC Programms modifizieren:

```
130 POKE 49201,3*16+4
```

Ist diese Änderung erfolgt, so erscheint der Kreis in der Farbe türkis (CYN Farbwert 3) und die Ellipse in der Farbe purpur (PUR Farbwert 4).

5. Sprites

Sprite, wörtlich ins Deutsche übersetzt, heißt soviel wie Geist oder Kobold. Diese Bezeichnung ist nicht ganz unzutreffend, denn ein Sprite ist ein freiprogrammierbares Gebilde, das aus 504 Einzelpunkten besteht. Von den 504 Punkten stehen uns 24 Punkte in horizontaler Richtung und 21 Zeilen in vertikaler Richtung zur Verfügung. Der Vorteil dieser Sprites ist, daß man sie sehr einfach über den gesamten Bildschirm bewegen kann und ihre Farbe frei wählbar ist. Auch mehrfarbige Sprites sind möeglich; diese werden im Kapitel Multi Color Sprites besprochen. Sprites können in X und Y Richtung vergrössert werden. Ferner kann man Kollisionen überprüfen, zum einen die Kollision der Sprites untereinander und zum anderen die Kollisionen der Sprites mit dem Hintergrund. Insgesamt lassen sich grundsätzlich mit dem Commodore 64 acht Sprites gleichzeitig darstellen. Mit einigen Tricks ist es aber auch möglich, mehr als acht Sprites darzustellen, dazu benötigt man das sogenannte Interrupt Raster Register. Überlappen sich Sprites mit dem Hintergrund, so gibt es Prioritäten, die programmiert werden können, das heißt, welche Sprites Priorität vor dem Hintergrund haben und umgekehrt. Alle Spritefunktionen laufen in allen Grafikmodi, wie zum Beispiel im Standard Bit Map Mode.

5.1 Standard Sprites

Unter einem Standard Sprite versteht man ein einfarbiges Sprite, das heißt, daß alle gesetzten Punkte die gleiche Farbe besitzen.
Als erstes werden wir uns anschauen, wie man ein Standard Sprite definiert. Wie schon gesagt, besteht ein Sprite aus 24 x 21 = 504 Punkten. Einen Punkt können wir mit einem Bit ansteuern,

das heißt, daß wir 504 Bit benötigen. 504 Bits entsprechen 63 Byte. Insgesamt brauchen wir 63 Byte für die Bit Map eines Sprites, die festlegt, wie ein Sprite aussieht.
Horizontal, in X-Richtung, verfügen wir ueber 24 Punkte. Um diese 24 Punkte ansteuern zu können benötigen wir 24 Bit = 3 Byte. Eine Spritedefinition sieht damit etwa so aus:

Spritedefinition:

Zeile	Byte 1	Byte 2	Byte 3
1	XXXX XXXX	XXXX XXXX	XXXX XXXX
2	XXXX XXXX	XXXX XXXX	XXXX XXXX
3	...	...	...
...	...	...	...
20	XXXX XXXX	XXXX XXXX	XXXX XXXX
21	XXXX XXXX	XXXX XXXX	XXXX XXXX

Um nun ein Sprite eindeutig definieren zu können müssen wir dem Computer mitteilen wo die 63 Byte der Spritedefinition im Speicherbereich stehen. Diese Information geben wir in den sogenannten Sprite Pointer ein. Im Sprite Pointer (jedes Sprite hat einen eigenen Pointer) steht die Startadresse der Spritedefinition dividiert durch 64. Es muß also der 64er Block angegeben werden. Diese Angabe wird für jedes Sprite einzeln ab Adresse 2040 ($07F8) eingespeichert. Sprite 0 besitzt den Sprite Pointer in 2040 ($07F8), Sprite 1 den Pointer in 2041 ($07F9) usw. Nehmen wir nun an, daß sich unsere Spritedefinition im Bereich von Adresse 832 ($0340) bis 895 ($037F) befindet. Arbeiten wir mit Sprite 0, so muß im Sprite Pointer für Sprite 0 der Wert 832 / 64 = 13 stehen, also POKE 2040,13.
Arbeiten wir in einem anderen 16k Bereich, der vom Standardwert abweicht, so muß die Startadresse

dieses neuen Bereiches zu den Adressen der Sprite Pointer hinzuaddiert werden.

Sprite Pointer Definition:

Sprite Nr.	Adresse des Sprite Pointers
0	2040 ($07F8)
1	2041 ($07F9)
2	2042 ($07FA)
3	2043 ($07FB)
4	2044 ($07FC)
5	2045 ($07FD)
6	2046 ($07FE)
7	2047 ($07FF)

Damit wir jetzt ein definiertes Sprite auf dem Bildschirm sehen, müssen wir die X und Y Koordinaten des Sprites bestimmen und die Farbe setzen. Schließlich ist das betreffende Sprite in Adresse 53269 ($D015) einzuschalten. In der nachfolgenden Abbildung ist das für ein Sprite zutreffende Bit durch ein X gekennzeichnet. Unwichtige erhalten ein -.

Einschalten und Farbe der Sprites:

Sprite Nr.	Farbe	Ein- und Ausschalten in 53269 ($D015)		
0	53287 ($D027)	---- ---X	W=1	($01)
1	53288 ($D028)	---- --X-	W=2	($02)
2	53289 ($D029)	---- -X--	W=4	($04)
3	53290 ($D02A)	---- X---	W=8	($08)
4	53291 ($D02B)	---X ----	W=16	($10)
5	53292 ($D02C)	--X- ----	W=32	($20)
6	53293 ($D02D)	-X-- ----	W=64	($40)
7	53294 ($D02E)	X--- ----	W=128	($80)

Wollten wir zum Beispiel Sprite 0 einschalten, so verwenden wir folgende Zeile:

BASIC:

```
POKE 53269,PEEK (53269) OR W
; Sprite n einschalten, W aus Tabelle
```

MASCHINENSPRACHE:

```
    SPREIN =$D015
LDA SPREIN     ; Wert laden
ORA #$#W       ; entsprechende Bits setzen
STA SPREIN     ; Sprite n ein, W aus Tabelle
RTS
```

X Koordinaten der Sprites:

Sprite Nr.(N)	X (Low)	X (High) in 53264 ($D010)	Wert W
0	53248 ($D000)	---- ---X	1 ($01)
1	53250 ($D002)	---- --X-	2 ($02)
2	53252 ($D004)	---- -X--	4 ($04)
3	53254 ($D006)	---- X---	8 ($08)
4	53256 ($D008)	---X ----	16 ($10)
5	53258 ($D00A)	--X- ----	32 ($20)
6	53260 ($D00C)	-X-- ----	64 ($40)
7	53262 ($D00E)	X--- ----	128 ($80)

Beträgt die X Koordinate mehr als 255, so muß das dem Sprite entsprechende Bit in der Adresse 53264 ($D010) gesetzt werden:

BASIC:

```
POKE 53264,PEEK (53264) OR W
; setzen des X-High Bits, W aus Tabelle
```

MASCHINENSPRACHE:

```
    XHIGH =$D010
LDA XHIGH   ; Wert laden
ORA #$W     ; entsprechende Bits setzen
STA XHIGH   ; X-High Bit setzen, W aus Tabelle
RTS
```

Y Koordinaten der Sprites:

Sprite Nr.	Y Koordinate
0	53249 ($D001)
1	53251 ($D003)
2	53253 ($D005)
3	53255 ($D007)
4	53257 ($D009)
5	53259 ($D00B)
6	53261 ($D00D)
7	53263 ($D00F)

Als nächstes sind die entsprechenden Bits für die Spritevergrößerung in X- und in Y-Richtung angegeben:

Spritevergrößerung:

Sprite Nr.	X-Vergröß. 53277($D01D)	Y-Vergröß. 53271($D017)	Wert W
0	---- ---X	---- ---X	1 ($01)

1	---- --X-	---- --X-	2 ($02)
2	---- -X--	---- -X--	4 ($04)
3	---- X---	---- X---	8 ($08)
4	---X ----	---X ----	16 ($10)
5	--X- ----	--X- ----	32 ($20)
6	-X-- ----	-X-- ----	64 ($40)
7	X--- ----	X--- ----	128 ($80)

BASIC:

```
POKE 53277,PEEK (53277) OR W
; setzt X-Vergrößerung von Sprite n
POKE 53271,PEEK (53271) OR W
; setzt Y-Vergrößerung von Sprite n
```

MASCHINENSPRACHE:

```
    XVERGR =$D01D
LDA XVERGR  ; Wert laden
ORA #$W     ; entsprechende Bits setzen
STA XVERGR  ; setzt X-Vergrößerung von Sprite n
RTS

    YVERGR =$D017
LDA YVERGR  ; Wert laden
ORA #$W     ; entsprechende Bits setzen
STA YVERGR  ; setzt Y-Vergrößerung von Sprite n
RTS
```

Wir werden nun besprechen, welches Sprite gegenüber den anderen Sprites Priotität hat. Das Sprite mit der niedriegsten Nummer hat die höchste Priorität. Wenn sich beispielsweise Sprite 0, Sprite 1 und Sprite 7 in einigen Punkten überschneiden, ist Sprite 0 im Vordergrund, danach kommt Sprite 1 und Sprite Nummer 7 hat die niedrigste Priorität.
Gegenüber dem Hintergrund können wir die

Priorität im Register 53275 ($D01B) bestimmen. Hat beispielsweise Sprite 0 vor dem Hintergrund Priorität, so ist in Adresse 53275 ($D01B) das Bit 0 zu löschen. Soll der Hintergrund vor dem Sprite 0 erscheinen, so muß Bit 0 auf 1 sein.
Die Tabelle auf der nächsten Seite verdeutlicht nochmals den Zusammenhang zwischen der Priorität Sprite-Hintergrund.

Hintergrund Sprite Priorität:

Srite Nr.	Spr. vor Hintergr. 53275 ($D01B)	Hintegr. vor Spr. 53275 ($D01B)
0	---- ---0	---- ---1
1	---- --0-	---- --1-
2	---- -0--	---- -1--
3	---- 0---	---- 1---
4	---0 ----	---1 ----
5	--0- ----	--1- ----
6	-0-- ----	-1-- ----
7	0--- ----	1--- ----

BASIC:

```
POKE 53275,PEEK (53275) OR 8
; Sprite 3 vor Hintergrund setzen
```

MASCHINENSPRACHE:

```
    HISPR =$D01B
LDA HISPR    ; Wert laden
ORA #$08     ; Bit 3 setzen
STA HISPR    ; setzt Hintergrund vor Sprite 3
RTS
```

Bei der Sprite-Sprite Kollision sind in Adresse

53278 ($D01E) die den Sprites entsprechenden Bits gesetzt, die in die Kollision verwickelt sind. Hier entspricht Bit 0 dem Sprite 0, Bit 1 dem Sprite 1 usw. Ähnliches gilt für die Sprite-Hintergrund-Kollision. Das Register, in dem die Kollision abgefragt werden kann, ist 53279 ($D01F). Die Bits der Sprites, die mit dem Hintergrund kollidieren, werden gesetzt.
Diese beidn Kollisionsregister haben eine Besonderheit. Sie behalten auch nach der Kollision zwischen Sprite-Sprite oder Sprite-Hintergrund ihren Wert so lange, bis sie zum Beispiel mit PEEK gelesen werden. Damit wird auch bei sehr kurzen Kollisionen eine Verfügbarkeit des Wertes sichergestellt.
Besprechen wir nun noch, wie man einzelne Bits setzen bzw. löschen kann:

Setzen einzelner Bits:

BASIC:

```
POKE Adresse,PEEK (Adresse) OR W
```

Beispiel: Sollen von einem Byte die mit X gekennzeichneten Bits gesetzt werden, so beträgt der Wert W:

```
--X- --XX  W=0*128+0*64+1*32+0*16+0*8-0*4+1*2+1*1
           W=35 ($23)
```

MASCHINENSPRACHE:

```
LDA Adresse ; Wert laden
ORA #$W     ; entsprechende Bits setzen
STA Adresse ; abspeichern
RTS
```

Löschen einzelner Bits:

BASIC:

POKE Adresse,PEEK (Adresse) AND W

Beispiel: Sollen von einem Byte die mit X gekennzeichneten Bits gelöscht werden, so beträgt der Wert W:

```
XX-- X-X- W = 255 - (1*128 + 1*64 + 1*8 - 1*2)
          W = 255 - 202
          W = 53 ($35)
```

MASCHINENSPRACHE:

```
LDA Adresse ; Wert laden
AND #$W     ; entsprechende Bits löschen
STA Adresse ; abspeichern
RTS
```

Zum Schluß des Standard Sprite Kapitels sei noch ein Beispielprogramm angegeben, das einige der zuvor besprochenen Zustände enthält:

BASIC:

```
100 REM STANDARD SPRITES
110 FORI=832TO895
120 READA
130 POKEI,A
140 NEXT
145 PRINT"[illegible]
150 POKE53280,2:POKE53281,11
160 POKE2040,13:POKE2041,13
```

```
170 POKE53288,5
175 POKE53275,PEEK(53275)OR1
180 POKE53269,PEEK(53269)OR3
190 FORI=0TO2*πSTEP.1
200 X1=100-50*SIN(I):X2=100-50*COS(I)
210 Y1=100-50*COS(I):Y2=100-50*SIN(I)
220 POKE53248,X1:POKE53249,Y1
225 POKE53250,X2:POKE53251,Y2
227 IFPEEK(53279)=1THENF=F+1
230 POKE53287,F:NEXT:GOTO190
1000 DATA 7,231,224,24,24,24,48,60,12,96,102,6
1010 DATA192,195,3,199,231,225,152,153,25,176
1020 DATA189,13,224,221,7,128,131,3,128,129,1
1030 DATA192,195,3,224,231,7,176,189,13,140
1040 DATA141,25,135,231,225,192,195,3,96,102,6
1050 DATA48,60,12,24,24,24,7,231,224
1060 DATA 7,231,224,24,24,24,48,60,12,96,102,6
10000 DATA 7,231,224,24,24,24,48,60,12,96,102,6
```

Erläuterungen zum BASIC Programm:

110: Schleife zum Einlesen der Spritedaten ab Adresse 832 (64-Block Nr.13)
140: Schleifenende
145: Hintergrundinformation auf Bildschirm
150: Rahmenfarbe rot (Farbwert 2), Hintergrundfarbe grau (Farbwert 11)
160: Sprite Pointer für Sprite 0 und Sprite 1 auf Block 13 ab Adresse 13 * 64 = 832 setzen
170: Farbe für Sprite 1 ist purpur (Farbwert 5)
175: Hintergrund hat Priorität vor Sprite 0
180: Sprite 0 und Sprite 1 einschalten
190: Schleife für die Kreisbahn der Sprites
200: Mathematische Berechnung der X Koordinaten
210: Mathematische Berechnung der Y Koordinaten
220: Koordinaten von Sprites 0 einspeichern
225: Koordinaten von Sprites 1 einspeichern
227: Auf Kollision Sprite 0 mit Hintergrund abfragen; liegt eine Kollision vor, dann Farb-

wert von Sprite 0 um eins erhöhen
230: Farbe von Sprite 0 einspeichern; Schleifenende; Sprung zur Zeile 190
1000: DATA Zeilen für die Spritedefinition

Das Programm zeichnet zwei Sprites, die entgegengesetzt eine Kreisbahn beschreiben. Kollidiert ein Sprite mit dem Hintergrund, so ändert es seine Farbe. Ein Sprite hat vor dem Hintergrund Priorität, das andere nicht. Sprite 0 hat Priorität vor Sprite 1.

5.2 Multi Color Sprites

Im Gegensatz zu den einfarbigen Standard Sprites können die Multi Color Sprites aus insgesamt vier Farben bestehen. Eine Farbe der vier ist transparent, daß heißt, diese Punkte nehmen dieselbe Farbe wie der Hintergrund an. Wir können also über drei Farben verfügen, die sich vom Hintergrund unterscheiden. Dabei sind zwei definierte Farbwerte für alle Sprites gleich, während ein Farbwert für jedes Sprite individuell programmiert werden kann. Kontrolliert wird der Sprite Multi Color Mode durch Adresse 53276 ($D01C). Jedes Bit dieses Registers entspricht einem Sprite. Ist ein Bit gesetzt, so befindet sich das entsprechende Sprite im Multi Color Mode. Die Definition der verschiedenfarbigen Punkte bringt es mit sich, daß sich die Auflösung des Sprites in X-Richtung halbiert. Es stehen demnach horizontal 12 Punkte, genauer gesagt Doppelpunkte, und vertikal (in Y-Richtung) weiterhin 21 Zeilen zur Verfügung. Die Farbinformation (also woher die Farbe eines Punktes stammt) wird wie üblich durch zwei Bits bestimmt:

Bits	Farbinformation kommt aus	Adresse
00	Hintergrundfarbregister 0	53281 ($D021)
01	Multi Color Register 0	53285 ($D025)
10	Sprite Color Register	ab 53287 ($D027)
11	Multi Color Register 1	53286 ($D026)

Ist das Bitmuster 10, so kommt die Farbinformation aus dem Sprite Color Register. Das Register ist dasselbe, aus dem die Farbe der Standard Sprites genommen wird. Für jedes Sprite kann somit eine eigene dritte Farbe definiert werden.

Sprite Multi Coler Mode (SMCM):

Sprite Nr.	SMCM ein 53276 ($D01C)	SMCM aus 53276 ($D01C)
0	---- ---1	---- ---0
1	---- --1-	---- --0-
2	---- -1--	---- -0--
3	---- 1---	---- 0---
4	---1 ----	---0 ----
5	--1- ----	--0- ----
6	-1-- ----	-0-- ----
7	1--- ----	0--- ----

BASIC:

```
POKE 53276,PEEK (53276) OR 1
; Sprite 1 ist ein Multi Color Sprite
```

MASCHINENSPRACHE:

```
    SMCM =$D01C
LDA SMCM     ; Wert laden
ORA #$01     ; Bit 0 setzen
```

```
STA SMCM     ; Sprite 0 ist ein Multi Color Sprite
RTS
```

BASIC:

```
POKE 53276,PEEK (53276) AND 254
; Sprite 0 ist ein Standard Sprite
```

MASCHINENSPRACHE:

```
    SMCM =$D01C
LDA SMCM     ; Wert laden
AND #$FE     ; Bit 0 löschen
STA SMCM     ; Sprite 0 ist ein Standard Sprite
RTS
```

Als Beispiel für ein Multi Color Sprite wird im nachfolgenden Programm ein dreifarbiges Sprite erstellt. Dieses Sprite ist in X und Y Richtung vergrössert. Die drei Farben der als Sprite nachgebildeten deutschen Fahne, sind schwarz (Farbwert 0), rot (Farbwert 2) und gelb (Farbwert 7), alles auf grauem Hintergrund (Farbwert 11).

BASIC:

```
100 REM MULTI COLOR SPRITE
110 PRINT"[illegible]";
115 FORI=832TO895:READA:POKEI,A:NEXT
120 POKE53280,2:POKE53281,11
130 POKE53285,0
140 POKE53286,7
150 POKE53287,2
155 POKE2040,13
160 POKE53276,PEEK(53276)OR1
```

```
170 POKE53248,100:POKE53249,100
180 POKE53269,PEEK(53269)OR1
190 POKE53271,PEEK(53271)OR1
200 POKE53277,PEEK(53277)OR1
1000 DATA85,85,85,85,85,85,85,85,85
1010 DATA170,170,170,170,170,170,170,170,170
1020 DATA255,255,255,255,255,255,255,255,255
1030 DATA0,0,0,0,0,0,0,0,0
1040 DATA85,85,85,85,85,85,85,85,85
1050 DATA170,170,170,170,170,170,170,170,170
1060 DATA255,255,255,255,255,255,255,255,255
1070 DATA0
```

Erläuterungen zum BASIC Programm:

110: Bildschirm löschen
115: Schleife für das Einlesen der Spritedaten
120: Rahmenfarbe ist rot (Farbwert 2) und Hintergrundfarbe ist grau (Farbwert 11)
130: Farbe schwarz (Farbwert 0) in Multi Color Register 0
140: Farbe gelb (Farbwert 7) in Multi Color Register 1
150: Farbe rot (Farbwert 2) in Farbregister des Sprites 0
155: Sprite Pointer auf 832 (13 * 64 = 832) setzen
160: Sprite 0 ist Multi Color Sprite
170: Koordinaten für Sprite 0 setzen
180: Schaltet Sprite 0 ein
190: Y Vergrößerung Sprite 0 ein
200: X Vergrößerung Sprite 0 ein
1000: DATA Zeilen für die Spritedefinition

Im Sprite Multi Color Mode gelten dieselben zu erreichenden Zustände wie im Standard Sprite Mode. Diese sind die Vergrößerung und die Prioritätsbestimmung.

6. Sonstige Besonderheiten des Video Chips

Neben den bekannten Betriebsarten des Video Chips, wie Standard Bit Map Mode oder Multi Color Character Mode, gibt es noch einige andere Register, die weitere Moeglichkeiten eröffnen. Diese zusätzlichen Möglichkeiten zur vollen Nutzung des Video Chips möchte ich nun erläutern.

6.1 Smooth Scrolling

Smooth Scrolling bedeutet das sanftes Verschieben des gesamten Bildschirms. Der Video Chip übernimmt dabei die Aufgabe, den Bildschirm in alle Richtungen um Einzelpunkte zu verschieben. Man gibt hierbei nur die X- und Y-Koordinaten an, an die der Bildschirm gesetzt werden soll. Die beiden Koordinaten können Werte von 0...7 annehmen. Das bedeutet also, daß wir horizontal und vertikal über acht Positionen verfügen können. Übergeben wir die Koordinaten dem Computer, so wird der gesamte Bildschirm an die eingegebene Stelle verschoben. Soll aber der Bildschirm um mehr als acht Einzelpunkte in eine Richtung verschoben werden, so müssen Sie die 8 x 8 Gruppen von Punkten an eine neue Stelle setzen und die Position des Bildschirms erneut auf den zutreffenden Wert setzen. Dies vereinfacht beispielsweise das Verschieben der Bit Map im Standard- oder Multi Color Bit Map Mode sehr. Gäbe es diese Möglichkeit nicht, so müßte man die 8000 Byte lange Bit Map in der hochauflösenden Grafik bitweise verschieben. Das ist zwar auch möglich, aber sehr umständlich und unübersichtlich.
Damit keine Störungen am Rand des Bildschirms auftreten, wechseln wir beim Smooth Scrolling das Bildschirmformat auf 24 Zeilen mit 38 Zeichen pro Zeile. Im Normalzustand beträgt das Bildschirmformat 25 Zeilen mit je 40 Zeichen pro Zeile. Diese Einschränkung des Bildschirms kann geson-

dert in X- und in Y-Richtung durchgeführt werden. Für unseren Zweck verkleinern wir den Bildschirm in beiden Richtungen. Kontrolliert wird das Bildschirmformat durch Bit 3 in Adresse 53270 ($D016) fuer die X-Einschränkung und Bit 3 der Adresse 53265 ($D011) für die Y-Verkleinerung.

BASIC:

```
POKE 53270,PEEK (53270) AND 247
; setzt Bildschirmformat auf 38 Zeichen pro Zeile
POKE 53265,PEEK (53265) AND 247
; setzt Bildschirmformat auf 24 Zeilen
```

MASCHINENSPRACHE:

```
    XFORM =$D016
    YFORM =$D011
LDA XFORM   ; Wert laden
AND #$F7    ; Bit 3 löschen
STA XFORM   ; 38 Zeichen pro Zeile setzen
LDA YFORM   ; Wert laden
AND #$F7    ; Bit 3 löschen
STA YFORM   ; 24 Zeilen setzen
RTS
```

Die acht X- und Y-Positionen des Bildschirms werden durch Bit 0...3 der Adressen 53270 ($D016) und 53265 ($D011) kontrolliert und zwar bestimmt 53270 ($D016) die X-Position und 53265 ($D011) die Y-Position des Bildschirms. Soll nun der Bildschirm sanft, daß heißt um je einen Einzelpunkt verschoben werden, so ändert man einfach kontinuierlich den X-Wert der Bildschirmposition. Auf der nächsten Seite sind die beiden Befehle zur Ansteuerung der erwünschten Bildschirmposition angegeben. Beide Koordinaten können Werte von 0...7 annehmen.

BASIC:

```
POKE 53270,( PEEK (53270) AND 248) OR X
; X Koordinate der Bildschirmposition
POKE 53265,( PEEK (53265) AND 248) OR Y
; Y Koordinate der Bildschirmposition
```

MASCHINENSPRACHE:

```
    XFORM =$D016
    YFORM =$D011
LDA XFORM    ; Wert laden
AND #$F8     ; Bit 0...3 löschen
ORA #$X      ; entsprechende Bits setzen
STA XFORM    ; X-Koordinate der Bildschirmposition
LDA YFORM    ; Wert laden
AND #$F8     ; Bit 0...3 löschen
ORA #$Y      ; entsprechende Bits setzen
STA YFORM    ; Y-Koordinate der Bildschirmposition
RTS
```

Besprechen wir nun ein einfaches Beispiel, das unseren Bildschirm sanft verschiebt:

BASIC:

```
100 REM SMOOTH SCROLLING
110 POKE53265,PEEK(53265)AND247
120 PRINT"[illegible]"
130 POKE53265,(PEEK(53265)AND248)+7:PRINT
140 PRINT"[illegible]      BEISPIEL:  SMOOTH SCROLLING";
150 FORP=6TO0STEP-1
160 POKE53265,(PEEK(53265)AND248)+P
170 D=1↑1↑1
180 NEXT:GOTO130
```

Erläuterungen zum BASIC Programm:

110: Bildschirmformat auf 24 Zeilen einschränken
120: Cursor an den unteren Bildschirmrand setzen
130: Position des Bildschirms auf den ersten Wert für das Verschieben setzen; hier: die Y-Koordinate
140: Beispieltext auf den Bildschirm schreiben
150: Schleife für das Smooth Scrolling setzen
160: Y-Koordinate kontinuierlich ändern für das Verschieben in Y-Richtung
170: Verzögerung
180: Ende Schleife und erneut beginnen

Das Programm schränkt den Bildschirm in Y-Richtung ein und schreibt einen Beispieltext, der dann im Smooth Scrolling Modus sanft von unten nach oben verschoben wird. Wenn man nicht das serienmäßig eingebaute Verschieben einer Zeile bei Erreichen des unteren Bildschirmrandes mit dem Cursor benutzt, so muß man am besten mittels eines kleinen Maschinenprogramms die neue, unsichtbare Zeile setzen. Die Zeile ist deswegen unsichtbar, weil sie durch den eingeschränkten Bildschirm in den unsichtbaren Bereich geschrieben wird. Das Programm muß dann nur immer eine 8 x 8 Matrix auf einmal verschieben. Das restliche Verschieben erledigt dann der Video Chip selbständig mit den beiden Smooth Scrolling Registern.

6.2 Screen Blanking

Das sogenannte Screen Blanking bedeutet, daß der gesammte Bildschirm dieselbe Farbe annimmt. Die Farbe ist die Rahmenfarbe. Dieser Effekt tritt zum Beispiel beim Laden von Programmen von Kassette auf.
Kontrolliert wird das Screen Blanking durch Bit 4 der Adresse 53265 ($D011):

BASIC:

```
POKE 53265,PEEK (53265) AND 239
; Bildschirm aus
```

MASCHINENSPRACHE:

```
    BILDSCH =$D011
LDA BILDSCH ; Wert laden
AND #$EF    ; Bit 4 löschen
STA BILDSCH ; Bildschirm aus
RTS
```

BASIC:

```
POKE 53265,PEEK (53265) OR 16
; Bildschirm ein
```

MASCHINENSPRACHE:

```
    BILDSCH =$D011
LDA BILDSCH ; Wert laden
ORA #$10    ; Bit 4 setzen
STA BILDSCH ; Bildschirm ein
RTS
```

Während der Bildschirm 'aus' ist, daß heißt der gesamte Bildschirm hat die Farbe des Rahmens, gehen die Daten, die sich vorher auf dem Bildschirm befanden, nicht verloren.

6.3 Raster Register

Mit dem Rasterregister verfügen wir über ein Register, in dem die Zeile des Bildschirms steht, die gerade vom Strahl des Fernsehers durchlaufen

Die unteren 8 Bit, Bit 0...7, stehen in Adresse 53266 ($D012). Das höchste Bit, Bit 8 der Rasterzeile, entspricht Bit 7 der Adresse 53265 ($D011).
Mit diesem Register ist es zum Beispiel möglich, die obere Hälfte des Bildschirms für hochauflösende Grafik zu verwenden, während im unteren Teil normaler Text erscheint. An einem einfachen Beispiel sei hier die Funktion des Raster Registers erläutert.
Das nachfolgende Programm ändert in einem bestimmten Teil des Bildschirms Rahmen- und Hintergrundfarbe. Somit entsteht quer über den ganzen Bildschirm ein breiter roter Strich, während der andere Hintergrund gelb gewählt wurde. Das BASIC Programm liest mittels DATA Zeilen ein kleines Maschinenprogramm ein, das anschließend aufgerufen wird. Nachfolgend wird auch das Maschinenprogramm erläutert:

BASIC:

```
100 REM RASTER REGISTER
110 FORI=49152TO49185
120 READA
130 POKEI,A
140 NEXT
150 SYS49152
1000 DATA120,173,18,208,201,80,208,249,169,2
1010 DATA141,32,208,141,33,208,173,18,208,201
1020 DATA133,208,249,169,7,141,32,208,141,33
1030 DATA208,76,0,192
```

Auf der nächsten Seite ist das Maschinenprogramm abgedruckt, das den Streifen mit Hilfe des Raster Registers im Bildschirm erzeugt.

MASCHINENSPRACHE:

```
C000 78          SEI
C001 AD 12 D0    LDA $D012
C004 C9 50       CMP #$50
C006 D0 F9       BNE $C001
C008 A9 02       LDA #$02
C00A 8D 20 D0    STA $D020
C00D 8D 21 D0    STA $D021
C010 AD 12 D0    LDA $D012
C013 C9 85       CMP #$85
C015 D0 F9       BNE $C010
C017 A9 07       LDA #$07
C019 8D 20 D0    STA $D020
C01C 8D 21 D0    STA $D021
C01F 4C 00 C0    JMP $C000
```

Erläuterungen zum Maschinenprogramm:

C000: Keinen Interrupt mehr ausführen um Störungen durch die Tastatur auszuschalten
C004: Rasterzeile mit Wert 80 ($50) vergleichen
C006: Solange warten bis der Wert 80 ($50) erreicht ist
C00A: Rahmenfarbe ist rot (Farbwert 2)
C00D: Hintergrundfarbe ist rot
C013: Rasterzeile mit Wert 133 ($85) vergleichen
C015: Solange warten bis Wert 133 ($85) erreicht ist
C019: Rahmenfarbe ist gelb (Farbwert 7)
C01C: Hintergrundfarbe ist gelb
C01F: Sprung zum Programmanfang

Unterbrochen werden kann das Programm nur mit der RUN/STOP RESTORE Funktion.

Das Raster Register erlaubt aber noch weitere Betriebsarten. Speichern wir eine Raster Zeile in das Register ein, so bleibt diese für den Video Chip erhalten. Jedesmal wenn nun die aktuelle Rasterzeile mit der eingespeicherten Zeile übereinstimmt wird Bit 7 im Interrupt Status Register gesetzt. Das Interrupt Status Register befindet sich in Adresse 53273 ($D019).

6.4 Weitere Register

Der Video Chip verfügt über vier weitere Register. Das erste ist das sogenannte Interrupt Status Register, in dem jede mögliche Art des Interrupts kontrolliert wird. Das andere ist das Interrupt Enable Register, mit dem man die im Interrupt Status Register angezeigten Interrupts sperren kann. Ferner gibt es noch zwei Register in Verbindung mit einem Light Pen (Stift zum Zeichnen auf dem Bildschirm).
Beginnen wir mit dem Interrupt Status Register. Das Register hat die Adresse 53273 ($D019). Die Bedeutung der verschiedenen Bits ist in nachfolgender Tabelle dargestellt:

Bit der Adresse 53273 ($D019)	Funktion
---- ---X	ist gesetzt, wenn die eingespeicherte Rasterzeile der aktuellen Zeile entspricht
---- --X-	ist gesetzt, wenn ein Sprite mit dem Hintergrund kollidiert
---- -X--	ist gesetzt, wenn ein Sprite mit einem anderen Sprite kollidiert
---- X---	ist bei negativen Impuls des Light Pens gesetzt

X--- ----	ist gesetzt, wenn eines der oben beschreibenen Bits gesetzt ist, daß heißt, wenn ein beliebiger Interrupt beginnt

Als nächstes sei das Interrupt Enable Register aufgeführt. Wie der Name schon sagt, kann man mit diesem Register etwas sperren. Und zwar können wir durch Setzen der entsprechenden Bits, Interrupts wie sie im Interrupt Status Register angezeigt sind, verhindern. Ist im Interrupt Enable Register zum Beispiel Bit 0 gesetzt, so wird der Vergleich mit der Rasterzeile gesperrt. Bit 1 entspricht dann Bit 1 usw. Das Interrupt Enable Register hat die Adresse 53274 ($D01A).
Diese Steuermöglichkeiten des Interrupts erlauben es, wie schon erwähnt, beispielsweise die obere Hälfte des Bildschirms im Standard Text Modus zu betreiben, während die untere Hälfte hochauflösende Grafik zeigt. Auch die Darstellung von mehr als acht Sprites zur gleichen Zeit ist mit diesen Interruptregistern möglich.
Beim Experimentieren mit Interrupts müssen Sie ausschließlich auf die Maschinensprache zurückgreifen, denn BASIC Routinen sind einfach zu langsam. Wenn Sie bedenken wie schnell Ihr Fernsehgerät oder Monitor ein Bild aufzeichnet, können Sie sich in etwa die Geschwindigkeit, mit der sich das Register der aktuellen Rasterzeile ändert, vorstellen, denn die Rasterzeile ist praktisch die Zeile auf dem Fernsehgerät oder Monitor, in der sich gerade der Strahl befindet, der das Bild erstellt.
Zwei Register sind noch in Verbindung mit dem Light Pen zu nennen. Wird am Eingang durch den Light Pen ein Signal ausgelöst, so können Sie die Koordinaten des Punktes auslesen, an dem sich der Light Pen auf dem Bildschirm befindet. Die X-Koordinate befindet sich in Register 53276 ($D013) und die Y-Koordinate in 53277 ($D014).

7. Anhang

7.1 IWT Sprite Komfort Kit

Der IWT SPRITE KOMFORT KIT, im folgenden kurz IWT-KIT genannt, ist ein hochwertiges 4k langes Maschinenprogramm und enthält ca. 40 neue Befehle und Funktionen, die in vielen Bereichen dem Programmierer das Bedienen des Computers erleichtern. Das Arbeiten mit den Sprites und der Umgang mit der hochauflösenden Grafik wird durch den IWT-KIT wesentlich vereinfacht. Einige Funktionen machen die Bedienung des Diskettenlaufwerks wesentlich komfortabler.
Des weiteren wird mit dem IWT-KIT eine Routine geliefert, die den hochauflösenden Grafikbildschirm in 8 x 8 Matrix auf dem Drucker ausgibt. Mit dem IWT-Spritegenerator, der ebenfalls mitgeliefert wird, ersparen Sie sich die umständliche Berechnung bei der Generierung von Sprites. Das Programm ist ein komfortabler Bildschirm Editor für Spritefiguren, die bei ihrer Erstellung gleichzeitig in normaler Größe und in Großdarstellung auf dem Bildschirm angezeigt werden. Nach Beendigung der Spriteerstellung können die Werte auf Floppy oder Band abgespeichert werden und stehen somit jederzeit zur Verfügung.
Zusätzlich zu den oben geschilderten Programmen sind auf der gelieferten Kassette oder Diskette verschiedene Demonstrationsprogramme enthalten, die die Anwendungen der IWT-KIT Funktionen erläutern.
Der IWT-KIT belegt keinen BASIC-Speicherplatz und alle Befehle laufen entweder in einem BASIC Programm oder im sogenannten Direkt Modus.
Angesteuert werden die IWT-KIT Befehle mit einem vorangehenden 'POUND' Zeichen.
In der folgenden Befehlsliste werden verschiedene Parameter verwendet. Diese können folgende Werte annehmen:

Parameterwerte:

a = 0...255
b = 0...255
h = $0000...$FFFF
x = 0...319
y = 0...199
n = 0...65535 (bei Spritebefehlen n = 1...8)
r = 1...99
g = Geräteadresse (1 = Band; 8 = Floppy)

IWT-KIT Befehlsliste:

Befehl	Bedeutung
£ Pn,a	Sprite-Pointer Zuweisung
£ In	Sprite ein
£ Kn,x,y	Angabe der Spritekoordinaten
£ Xn	X-Vergrößerung ein
£ Yn	Y-Vergrößerung ein
£ Vn	X-Vergrößerung wird aufgehoben
£ Wn	Y-Vergrößerung wird aufgehoben
£ Cn,a	Farbe des Sprites
£ Sn	Sprite vor Hintergrund
£ Hn	Hintergrund vor Sprite
£ On	Sprite aus
£ N	Alle Sprites aus
£ 5a,'NAME',g	Speichert Sprite ab
£ 6a,'NAME',g	Lädt Sprite
£ A	IWT-KIT aus
£ Fa,b	Farbe: Rahmen, Hintergrund
£ $n	Dezimal nach Hexadezimalumrechnung
£ Dh	Hexadezimal nach Dezimalumrechnung
£ R	Repeat alle Tasten ein/aus
£ £	Liest Directory von Disk ohne das vorher eingegebene Programm zu löschen
£ "	Floppybefehl allg. (OPEN 1,8,15...)
£ @	Floppyfehlerabfrage

ŧ 0	Löscht den Grafikbildschirm
ŧ 1	Hochauflösende Grafik ein
ŧ 2	Hochauflösende Grafik aus
ŧ La,b	Hintergrund-, Linien-, Punktfarbe
ŧ Qx,y	Setzt Punkt
ŧ Ux,y	Löscht Punkt
ŧ Ex1,y1,x2,y2	Zieht Linie zwischen den Endpunkten
ŧ Tx1,y1,x2,y2	Löscht Linie
ŧ Bx1,y1,x2,y2	Rahmen unter Angebe der Eckpunkte
ŧ Jx1,y1,x2,y2	Löscht Rahmen
ŧ Gx,y,r	Schreibt Kreis um x,y mit Radius r
ŧ Mx,y,r	Löscht Kreis
ŧ 3,'NAME',g	Speichert Grafikbildschirm ab
ŧ 4,'NAME',g	Lädt Grafikbildschirm
ŧ 7x,y,'TEXT'	Schreibt Text in Grafikbildschirm
ŧ 8x,y,'TEXT'	Schreibt Text revers

Da die Funktionen alle in Maschinensprache geschrieben sind, werden hohe Arbeitsgeschwindigkeiten erreicht. Der IWT SPRITE KOMFORT KIT besitzt die schnellste Grafikansteuerung, die derzeit auf dem Markt erhältlich ist. Eine Reihe von Beispielprogrammen, darunter auch ein Beispiel einer dreidimensionalen Grafik, verdeutlicht die Leistungsfähigkeit, die nicht zuletzt, durch die sorgfältig ausgedachten Maschinenroutinen und durch die kurze Erkennungszeit für die einzelnen Befehle ermöglicht wurde. Deswegen besitzen alle IWT-KIT Befehle einen Kennbuchstaben oder eine Kennziffer.
Den IWT SPRITE KOMFORT KIT können Sie auf Kassette oder Diskette beziehen. Bitte beachten Sie auch in diesem Zusammenhang die Anzeige in diesem Buch.
Im Lieferumfang ist selbsverständlich eine deutschsprachige Dokumentation enthalten, in der alle IWT SPRITE KOMFORT KIT Befehle ausführlich erläutert sind.

7.2 Video Chip Register

Adresse   Angesprochene Bits

53248 ($D000) XXXX XXXX Bit 0...7
X(Low)-Koordinate von Sprite 0

53249 ($D001) XXXX XXXX Bit 0...7
Y-Koordinate von Sprite 0

53250 ($D002) XXXX XXXX Bit 0...7
X(Low)-Koordinate von Sprite 1

53251 ($D003) XXXX XXXX Bit 0...7
Y-Koordinate von Sprite 1

53252 ($D004) XXXX XXXX Bit 0...7
X(Low)-Koordinate von Sprite 2

53253 ($D005) XXXX XXXX Bit 0...7
Y-Koordinate von Sprite 2

53254 ($D006) XXXX XXXX Bit 0...7
X(Low)-Koordinate von Sprite 3

53255 ($D007) XXXX XXXX Bit 0...7
Y-Koordinate von Sprite 3

53256 ($D008) XXXX XXXX Bit 0...7
X(Low)-Koordinate von Sprite 4

53257 ($D009) XXXX XXXX Bit 0...7
Y-Koordinate von Sprite 4

53258 ($D00A) XXXX XXXX Bit 0...7
X(Low)-Koordinate von Sprite 5

53259 ($D00B) XXXX XXXX Bit 0...7
Y-Koordinate von Sprite 5

```
53260 ($D00C) XXXX XXXX Bit 0...7
 X(Low)-Koordinate von Sprite 6

53261 ($D00D) XXXX XXXX Bit 0...7
 Y-Koordinate von Sprite 6

53262 ($D00E) XXXX XXXX Bit 0...7
 X(Low)-Koordinate von Sprite 7

53263 ($D00F) XXXX XXXX Bit 0...7
 Y-Koordinate von Sprite 7

53264 ($D010) ---- ---X Bit 0
 X(High)-Koordinate von Sprite 0
              ---- --X- Bit 1
 X(High)-Koordinate von Sprite 1
              ---- -X-- Bit 2
 X(High)-Koordinate von Sprite 2
              ---- X--- Bit 3
 X(High)-Koordinate von Sprite 3
              ---X ---- Bit 4
 X(High)-Koordinate von Sprite 4
              --X- ---- Bit 5
 X(High)-Koordinate von Sprite 5
              -X-- ---- Bit 6
 X(High)-Koordinate von Sprite 6
              X--- ---- Bit 7
 X(High)-Koordinate von Sprite 7

53265 ($D011) ---- -XXX Bit 0...2
 Anzahl der Rasterzeilen vom oberen Bildschirm-
 rand
              ---- X--- Bit 3
 38 Zeichen (Bit3=0)/40 Zeichen (Bit3=1)
              ---X ---- Bit 4
 Bildschirm aus (Bit4=0)/Bildschirm ein (Bit4=1)
              --X- ---- Bit 5
 Standard Bit Map Mode (Bit5=1)
              -X-- ---- Bit 6
```

X--- ---- Bit 7
höchstes Bit (Bit 8) der aktuellen Rasterzeile

53266 ($D012) XXXX XXXX Bit 0...7
Low Byte der aktuellen Nummer der Rasterzeile in der sich der Strahl befindet

53267 ($D013) XXXX XXXX Bit 0...7
X-Koordinate in der durch den Light Pen ein negatives Signal ausgelöst wurde

53268 ($D014) XXXX XXXX Bit 0...7
Y-Koordinate des Light Pens (vgl.53267 ($D013))

53269 ($D015) ---- ---X Bit 0
Sprite 0 ein (Bit0=1)
---- --X- Bit 1
Sprite 1 ein (Bit1=1)
---- -X-- Bit 2
Sprite 2 ein (Bit2=1)
---- X--- Bit 3
Sprite 3 ein (Bit3=1)
---X ---- Bit 4
Sprite 4 ein (Bit4=1)
--X- ---- Bit 5
Sprite 5 ein (Bit5=1)
-X-- ---- Bit 6
Sprite 6 ein (Bit6=1)
X--- ---- Bit 7
Sprite 7 ein (Bit7=1)

53270 ($D016) ---- -XXX Bit 0...2
Anzahl der Rasterpunkte vom linken Bildrand
---- X--- Bit 3
24 Zeilen (Bit3=0)/25 Zeilen (Bit3=1)
---X ---- Bit 4
Multi Color Bit Map Mode (Bit4=1)

```
53271 ($D017) ---- ---X Bit 0
 Vergrößerung in X-Richtung Sprite 0 (Bit0=1)
              ---- --X- Bit 1
 Vergrößerung in X-Richtung Sprite 1 (Bit1=1)
              ---- -X-- Bit 2
 Vergrößerung in X-Richtung Sprite 2 (Bit2=1)
              ---- X--- Bit 3
 Vergrößerung in X-Richtung Sprite 3 (Bit3=1)
              ---X ---- Bit 4
 Vergrößerung in X-Richtung Sprite 4 (Bit4=1)
              --X- ---- Bit 5
 Vergrößerung in X-Richtung Sprite 5 (Bit5=1)
              -X-- ---- Bit 6
 Vergrößerung in X-Richtung Sprite 6 (Bit6=1)
              X--- ---- Bit 7
 Vergrößerung in X-Richtung Sprite 7 (Bit7=1)

53272 ($D018) ---- XXX- Bit 1...3
 Startadresse Zeichenbasis und zwar die Bits
 11...13 der Adresse: --XX X000 0000 0000
              XXXX ---- Bit 4...7
 Startadresse Video-RAM  und zwar die Bits
 10...13 der Adresse: --XX XX00 0000 0000

53273 ($D019) ---- ---X Bit 0
 ist 1, wenn die eingespeicherte Rasterzeile mit
  der aktuellen übereinstimmt
              ---- --X- Bit 1
 ist 1, wenn ein Sprite mit dem Hintergrund
 kollidiert
              ---- -X-- Bit 2
 ist 1, wenn Sprites kollidieren
              ---- X--- Bit 3
 ist 1, wenn ein negatives Signal vom Light Pen
 ausgelöst  wurde
              X--- ---- Bit 7
 ist 1, wenn eines der anderen Bits auf 1 ist

53274 ($D01A) X--- XXXX Bit 0...3,Bit 7
 ist eines der Bits auf 1 so ist die bitgleiche
```

Funktion im Register 53273 ($D019) gesperrt

```
53275 ($D01B) ---- ---X Bit 0
 Hintergrund hat Vorrang vor Sprite 0 (Bit0=1)
              ---- --X- Bit 1
 Hintergrund hat Vorrang vor Sprite 1 (Bit1=1)
              ---- -X-- Bit 2
 Hintergrund hat Vorrang vor Sprite 2 (Bit2=1)
              ---- X--- Bit 3
 Hintergrund hat Vorrang vor Sprite 3 (Bit3=1)
              ---X ---- Bit 4
 Hintergrund hat Vorrang vor Sprite 4 (Bit4=1)
              --X- ---- Bit 5
 Hintergrund hat Vorrang vor Sprite 5 (Bit5=1)
              -X-- ---- Bit 6
 Hintergrund hat Vorrang vor Sprite 6 (Bit6=1)
              X--- ---- Bit 7
 Hintergrund hat Vorrang vor Sprite 7 (Bit7=1)

53276 ($D01C) ---- ---X Bit 0
 Sprite 0 im Sprite Multi Color Mode (Bit0=1)
              ---- --X- Bit 1
 Sprite 1 im Sprite Multi Color Mode (Bit1=1)
              ---- -X-- Bit 2
 Sprite 2 im Sprite Multi Color Mode (Bit2=1)
              ---- X--- Bit 3
 Sprite 3 im Sprite Multi Color Mode (Bit3=1)
              ---X ---- Bit 4
 Sprite 4 im Sprite Multi Color Mode (Bit4=1)
              --X- ---- Bit 5
 Sprite 5 im Sprite Multi Color Mode (Bit5=1)
              -X-- ---- Bit 6
 Sprite 6 im Sprite Multi Color Mode (Bit6=1)
              X--- ---- Bit 7
 Sprite 7 im Sprite Multi Color Mode (Bit7=1)

53277 ($D01D) ---- ---X Bit 0
 Vergrößerung in Y-Richtung Sprite 0 (Bit0=1)
```

---- --X- Bit 1
Vergrößerung in Y-Richtung Sprite 1 (Bit1=1)
---- -X-- Bit 2
Vergrößerung in Y-Richtung Sprite 2 (Bit2=1)
---- X--- Bit 3
Vergrößerung in Y-Richtung Sprite 3 (Bit3=1)
---X ---- Bit 4
Vergrößerung in Y-Richtung Sprite 4 (Bit4=1)
--X- ---- Bit 5
Vergrößerung in Y-Richtung Sprite 5 (Bit5=1)
-X-- ---- Bit 6
Vergrößerung in Y-Richtung Sprite 6 (Bit6=1)
X--- ---- Bit 7
Vergrößerung in Y-Richtung Sprite 7 (Bit7=1)

53278 ($D01E) XXXX XXXX Bit 0...7
dasjenige Bit ist gesetzt dessen Sprite
in eine Sprite-Sprite Kollision verwickelt ist

53279 ($D01F) XXXX XXXX Bit 0...7
dasjenige Bit ist gesetzt dessen Sprite
in eine Sprite-Hintergrund Kollision verwickelt
ist

53280 ($D020) XXXX XXXX Bit 0...7
Rahmenfarbe

53281 ($D021) XXXX XXXX Bit 0...7
Hintergrundfarbregister 0 (Bildschirmfarbe)

53282 ($D022) XXXX XXXX Bit 0...7
Hintergrundfarbregister 1

53283 ($D023) XXXX XXXX Bit 0...7
Hintergrundfarbregister 2

53284 ($D024) XXXX XXXX Bit 0...7
Hintergrundfarbregister 3

53285 ($D025) XXXX XXXX Bit 0...7
Sprite Multi Color Register 0

53286 ($D026) XXXX XXXX Bit 0...7
Sprite Multi Color Register 1

53287 ($D027) XXXX XXXX Bit 0...7
Farbe Sprite 0

53288 ($D028) XXXX XXXX Bit 0...7
Farbe Sprite 1

53289 ($D029) XXXX XXXX Bit 0...7
Farbe Sprite 2

53290 ($D02A) XXXX XXXX Bit 0...7
Farbe Sprite 3

53291 ($D02B) XXXX XXXX Bit 0...7
Farbe Sprite 4

53292 ($D02C) XXXX XXXX Bit 0...7
Farbe Sprite 5

53293 ($D02D) XXXX XXXX Bit 0...7
Farbe Sprite 6

53294 ($D02E) XXXX XXXX Bit 0...7
Farbe Sprite 7

7.3 Maschinensprachebefehle

```
ADC #$xx     $69 ; Add memory to accumulator with
ADC $xx      $65   carry
ADC $xx,Y    $75 ; Addiere Wert zum Akkumulator
ADC $xxxx    $6D   mit Carry Bit
ADC $xxxx,X  $7D
ADC $xxxx,Y  $79
ADC ($xx,X)  $61
ADC ($xx),Y  $71

AND #$xx     $29 ; AND memory with accumulator
AND $xx      $25 ; AND Wert mit Akkumulator
AND $xx,X    $35
AND $xxxx    $2D
AND $xxxx,X  $3D
AND $xxxx,Y  $39
AND ($xx,X)  $21
AND ($xx),y  $31

ASL A        $0A ; Shift left one bit (memory or
ASL $xx      $06   accumulator)
ASL $xx,X    $16 ; Speicher oder Akkumulator ein
ASL $xxxx    $0E   Bit nach links schieben
ASL $xxxx,X  $1E

BCC rr       $90 ; Branch on Carry Clear
                 ; Sprung bei Carry nicht gesetzt

BCS rr       $B0 ; Branch on Carry Set
                 ; Sprung bei gesetztem Carry

BEQ rr       $F0 ; Branch on result zero
                 ; Sprung bei Ergebnis = 0

BIT $xx      $24 ; Test bits in memory with
BIT $xxxx    $2C   accumulator
                 ; Prüft  Bits im Speicher mit de
                   Akkumulator
```

```
BMI rr         $30 ; Branch on result minus
                   ; Sprung wenn Ergebnis negativ

BNE rr         $D0 ; Branch on result non zero
                   ; Sprung bei Ergebnis ²³ 0

BPL rr         $10 ; Branch on result plus
                   ; Sprung wenn Ergebnis positiv

BRK            $00 ; Force Break
                   ; Stop Befehl

BVC rr         $50 ; Branch on overflow clear
                   ; Sprung wenn Überlauf  Bit nicht
                     gesetzt ist

BVS rr         $70 ; Branch on overflow set
                   ; Sprung wenn Überlauf  Bit ge-
                     setzt ist

CLC            $18 ; Clear Carry Flag
                   ; Lösche  Carry Bit

CLD            $D8 ; Clear decimal mode
                   ; Dezimal Modus aus

CLI            $58 ; Clear interrupt disable bit
                   ; Interrupt Bit löschen

CLV            $B8 ; Clear overflow flag
                   ; Überlauf  Bit löschen

CMP #$xx       $C9 ; Compare memory and accumulator
CMP $xx        $C5 ; Vergleicht Wert mit Akkumulator
CMP $xx,X      $D5
CMP $xxxx      $CD
CMP $xxxx,X    $DD
CMP $xxxx,Y    $D9
CMP ($xx,X)    $C1
CMP ($xx),Y    $D1
```

```
CPX #$xx     $E0 ; Compare memory and X
CPX $xx      $E4 ; Vergleicht Wert mit X-Register
CPX $xxxx    $EC

CPY #$xx     $C0 ; Compare memory and Y
CPY $xx      $C4 ; Vergleicht Wert mit Y-Register
CPY $xxxx    $CC

DEC $xx      $C6 ; Decrement memory by one
DEC $xx,X    $D6 ; Speicherwert -1
DEC $xxxx    $CE
DEC $xxxx,X  $DE

DEX          $CA ; Decrement X by one
                 ; X-Register -1

DEY          $88 ; Decrement Y by one
                 ; Y-Register -1

EOR #$xx     $49 ; Exclusive-Or memory with
EOR $xx      $45    accumulator
EOR $xx,X    $55 ; Exklusiv-Oder Wert mit Akkumu-
EOR $xxxx    $4D   lator
EOR $xxxx,X  $5D
EOR $xxxx,Y  $59
EOR ($xx,X)  $41
EOR ($xx),Y  $51

INC $xx      $E6 ; Increment memory by one
INC $xx,X    $F6 ; Speicherwert +1
INC $xxxx    $EE
INC $xxxx,X  $FE

INX          $E8 ; Increment X by one
                 ; X-Register +1

INY          $C8 ; Increment Y by one
                 ; Y-Register +1
```

```
JMP $xxxx    $4C ; Jump to new location
JMP ($xxxx)  $6C ; Sprung an neue Adresse

JSR $xxxx    $20 ; Jump to subroutine and returns
                 ; Sprung in Unterprogramm

LDA #$xx     $A9 ; Load accumulator with memory
LDA $xx      $A5 ; Akkumulator mit Wert laden
LDA $xx,X    $B5
LDA $xxxx    $AD
LDA $xxxx,X  $BD
LDA $xxxx,Y  $B9
LDA ($xx,X)  $A1
LDA ($xx),Y  $B1

LDX #$xx     $A2 ; Load X with memory
LDX $xx      $A6 ; X-Register mit Wert laden
LDX $xx,Y    $B6
LDX $xxxx    $AE
LDX $xxxx,Y  $BE

LDY #$xx     $A0 ; Load Y with memory
LDY $xx      $A4 ; Y-Register mit Wert laden
LDY $xx,X    $B4
LDY $xxxx    $AC
LDY $xxxx,X  $BC

LSR A        $4A ; Shift right one bit
LSR $xx      $46 ; ein Bit nach rechts schieben
LSR $xx,X    $56
LSR $xxxx    $4E
LSR $xxxx,X  $5E

NOP          $EA ; No operation
                 ; Keine Operation

ORA #$xx     $09 ; Or memory with accumulator
ORA $xx      $05 ; ODER Wert mit Akkumulator
ORA $xx,x    $15
ORA $xxxx    $0D
```

```
ORA $xxxx,X $1D
ORA $xxxx,Y $19
ORA ($xx,X) $01
ORA ($xx),Y $11

PHA           $48 ; Push accumulator on stack
                  ; Akkumulator auf Stapel schieben

PHP           $08 ; Push processer status on stack
                  ; Status auf Stapel schieben

PLA           $68 ; Pull accumulator from stack
                  ; Akkumulator vom Stapel holen

PLP           $28 ; Pull processor status from stack
                  ; Status vom Stapel holen

ROL A         $2A ; Rotate one bit left (memory or
ROL $xx       $26   accumulator)
ROL $xx,X     $36 ; ein Bit nach links rotieren
ROL $xxxx     $2E ; (Speicher oder Akkumulator)
ROL $xxxx,X   $3E

ROR A         $6A ; Rotate one bit right (memory or
ROR $xx       $66   accumulator)
ROR $xx,X     $76 ; ein Bit nach rechts rotieren
ROR $xxxx     $6E   (Speicher oder Akkumulator)
ROR $xxxx,X   $7E

RTI           $40 ; Return from Interrupt
                  ; Rückkehr  vom Interrupt

RTS           $60 ; Return from subroutine
                  ; Rücksprung  aus Unterprogramm

SBC #$xx      $E9 ; Subtract memory from accumulator
SBC $xx       $E5   with borrow
SBC $xx,X     $F5 ; Zieht Wert von Akkumulator mit
SBC $xxxx     $ED   Borrow ab
SBC $xxxx,X   $FD
```

```
SBC $xxxx,Y $F9
SBC ($xx,X) $E1
SBC ($xx),Y $F1

SEC         $38 ; Set carry flag
                ; Carry Bit setzen

SED         $F8 ; Set decimal mode
                ; Dezimal Modus ein

SEI         $78 ; Set interrupt disable status
                ; Interrupt Flag setzen

STA $xx     $85 ; Store accumulator in memory
STA $xx,X   $95 ; Akkumulator in Speicher
STA $xxxx   $8D   schreiben
STA $xxxx,X $9D
STA $xxxx,Y $99
STA ($xx,X) $81
STA ($xx),Y $91

STX $xx     $86 ; Store X in memory
STX $xx,Y   $96 ; X-Register in Speicher schrei-
STX $xxxx   $8E   ben

STY $xx     $84 ; Store Y in memory
STY $xx,X   $94 ; Y-Register in Speicher schrei-
                  ben

TAX         $AA ; Transfer accumulator to X
                ; Akkumulator nach X bringen

TAY         $A8 ; Transfer accumulator to Y
                ; Akkumulator nach Y bringen
TSX         $BA ; Transfer Stack Pointer to X
                ; Stapelzeiger nach X bringen
TXA         $8A ; Transfer X to accumulator
                ; X nach Akkumulator bringen
TXS         $9A ; Transfer X to stack pointer
                ; X in den Stapelzeiger bringen
```

```
TYA          $98 ; Transfer Y to accumulator
                 ; Y nach Akkumulator bringen
```

Parameterwerte in der Liste:

```
#$xx        Wert von ($00-$FF)
$xx          Speicherstelle in der Zero Page
$xxxx        Speicherstelle
rr           relative Adresse
```

Stichwortverzeichnis:

A

B

C

D

E

P

R

S

W

Z



Eine Hilfestellung für wirtschaftliche Entscheidungen sind Programmsammlungen, die die guten Grafik- und Farbmöglichkeiten des Computers nutzen. Diagramme, Sprites, optische Darstellungen von Simulationen werden eingesetzt, die die Ergebnisse verdeutlichen. Die finanzmathematischen Grundlagen sind zu jedem Programm beschrieben.

1983. 224 Seiten. Mit mehr. Abb. **Spiralh. DM 38,–/Fr. 38.–/S 342,–**
ISBN 3-88322-030-2



Bekanntlich verfügt der C 64 ›von Haus aus‹ über einen Baustein, der die Erzeugung von mehrstimmiger Musik erlaubt. Sowohl der Anfänger ohne musikalische Vorkenntnisse wird angesprochen, als auch der Musiker, der seine Ideen mit Hilfe des Computers umsetzen möchte.

In Vorb. Mai 1984. Ca. 200 Seiten. **Spiralh. Ca. DM 38,–/ca. Fr. 38.–/ca. S 342,–**
ISBN 3-88322-046-9



Dieses Buch führt nach einer Kurzbeschreibung der Grundlagen direkt in die Maschinensprachen-Programmierung ein, ohne erst die Befehle ›auf dem Trockenen‹ zu besprechen. Der Benutzer arbeitet sofort mit lauffähigen Programmen. Befehle werden dann eingeführt, wenn sie erforderlich sind.

In Vorb. April 1984. Ca. 260 Seiten. **Spiralh. Ca. DM 56,–/ca. Fr. 56.–/ca. S 498,–**
ISBN 3-88322-047-7



Programme reichen von stöchiometrischen Berechnungen bis zur Elementedatei. Besonders genutzt werden die grafischen Möglichkeiten. Die Anwendungsbeispiele gehen von Strukturformeln, grafischen Darstellungen von Versuchsanordnungen und Funktionen bis zur 3-D Grafik im Detail.

In Vorb. März 1984. Ca. 220 Seiten. **Spiralh. Ca. DM 48,–/ca. Fr. 48.–/ca. S 432,–**
ISBN 3-88322-049-3



Dieses Buch enthält eine ganze Reihe von sofort lauffähigen Spiel- und Simulationsprogrammen, möchte aber auch dazu anregen, diese Programme zu verändern und weiterzuentwickeln. Besonders reizvoll dürfte es wohl sein, den ›lernenden‹ Programmen noch etwas mehr ›Intelligenz‹ zu verleihen.

In Vorb. 1984. Ca. 200 Seiten. **Spiralh. Ca. DM 38,–/ca. Fr. 38.–/ca. S 342,–**
ISBN 3-88322-050-7





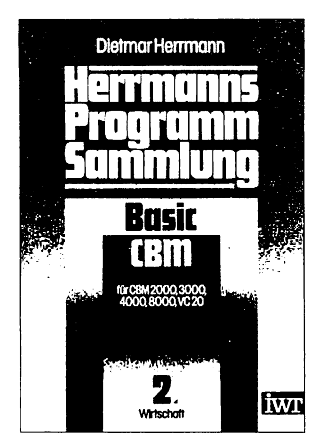
Grafikprogramme werden ›gehirngerecht‹ aufbereitet, d.h. man sieht, wie Grafikbefehle ›gehen‹: Neue Art des Formats – man bekommt ein ›Bild‹ des Befehls · Demo-Programme unterstützen das Gedächtnis · Bildschirm-Hardcopies als schnelles Nachschlagewerk · farbige Übersichtskarten zur Programmier-Erleichterung.

In Vorb. April 1984. Ca. 160 S. **Spiralh. Ca. DM 38,–/ca. Fr. 38.–/ca. S 342,–**
ISBN 3-88322-056-6



Der erste Band einer Reihe, die mit zahlreichen Programmen für Spiele und ›ernsthafte‹ Themen den Computer dem Benutzer näherbringt. Der Autor hat aus seiner Schulpraxis heraus Programme entwickelt, die das Lernen und Spielen mit dem Computer zum Vergnügen machen.

1983. 234 Seiten. **Kart. DM 32,–/Fr. 32.–/ S 288,–**
ISBN 3-88322-013-2



Jetzt wird es ernst: Hier wird Ihnen gezeigt, wie Sie mit dem Computer die Lohn- oder Einkommensteuererklärung erledigen, zeigt wohin die Staatsverschuldung geht – oder auch Ihre eigene, berechnet Ihre Zinsen (Soll oder Haben) auf der Bank, oder wie Sie Ihr Haus finanzieren können.

1983. 204 Seiten. **Kart. DM 32,–/Fr. 32.–/ S 288,–**
ISBN 3-88322-014-0



Dieses Buch enthält 40 mathematische Programme aus den Bereichen: Mehrregister-Arithmetik – Zahlentheorie – Kombinatorik – Algebra – Geometrie – numerische Mathematik. Neu ist die Langzahl-Arithmetik. Sie gestattet die Grundrechenarten für Zahlen bis 255 Stellen.

1983. 248 Seiten. **Kart. DM 42,–/Fr. 42.–/ S 378,–. ISBN 3-88322-016-7**



Der C 64 bietet vielseitige grafische Möglichkeiten. Dieses Buch gibt Informationen wie man Grafikfunktionen anwendet – Informationen, die man im Commodore-Handbuch nicht findet. Ausgehend von Grafiken mit den ›festen‹ Grafik-Zeichen wird systematisch zu den anspruchsvolleren Möglichkeiten, illustriert durch typische Beispiele, geführt.

1983. 138 S. 1 Folie. **Spiralh. DM 38,–/ Fr. 38.–/S 342,–. ISBN 3-88322-027-2**



Dieses Buch enthält 40 mathematische Programme aus den Bereichen: Mehrregister-Arithmetik – Zahlentheorie – Kombinatorik – Algebra – Geometrie – numerische Mathematik. Neu ist die Langzahl-Arithmetik. Sie gestattet die Grundrechenarten für Zahlen bis 255 Stellen.

1984. 260 Seiten. **Kart. DM 42,–/ Fr. 42.–/S 378,–**
ISBN 3-88322-048-5



Dieses Buch bietet eine systematische Einführung in die Programmiersprache BASIC. Außer vielen kleineren Programmen zur Illustrierung der BASIC-Anweisungen gibt es eine umfangreiche Programmsammlung zu den verschiedensten Themenbereichen. Die besonderen Fähigkeiten des C 64 werden mit vielen Programmbeispielen erläutert.

1983. 356 Seiten. **Spiralh. DM 56,–/ Fr. 56.–/S 498,–**
ISBN 3-88322-029-9

David Possin

CBASIC

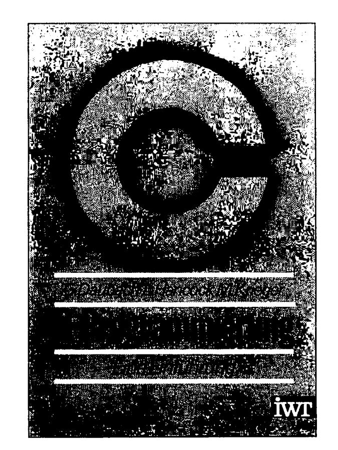
GSX : C-CP/M : DR-C

Einführung in die Verwendung von CBASIC für Grafik, Multitasking und Funktionsbibliotheken

iwt

Die Besonderheiten der 16-Bit Version von CBASIC werden erläutert. Spezielle Grafikbefehle sprechen das Betriebssystem GSX von DR direkt an. Weiter werden ›C‹-Funktionen mit CBASIC Programmen verknüpft, um so die Anwendung von Assembler zu umgehen. Vorteile von CP/M86, CCP/M86 und PC-DOS werden behandelt.

In Vorb. Juni 1984. Ca. 250 Seiten. **Spiralh. Ca. DM 68,–/ca. Fr. 68.–/ca. S 612,–. ISBN 3-88322-071-X**



Die Programmiersprache ›C‹ ist besonders zur Erstellung schneller, maschinennaher Programme geeignet, weist jedoch gegenüber der Assemblerprogrammierung wesentliche Vorteile auf, wie z. B. lokale und globale Variable, Prozeduren, Funktionen usw. Diese Einführung setzt beim Leser keine umfassenden Kenntnisse voraus.

In Vorb. 1984. Ca. 250 Seiten. **Geb. ca. DM 56,–/ca. Fr. 56.–/ca. S 498,– ISBN 3-88322-041-8**



Wer hat nicht bereits verzweifelt versucht, das ›Computerchinesenglisch‹ zu verstehen? Hier hilft das Wörterbuch der Computerei mit seinen über tausend Begriffen. Außerdem sind die wichtigsten Begriffe erklärt. Ein handliches Nachschlagewerk für jeden, der sich mit Computerei beschäftigt.

1983. 2., erw. Aufl. 144 Seiten. **Kart. DM 32,–/Fr. 32.–/S 288,– ISBN 3-88322-026-4**



Zwei leistungsfähige Programme, die unter CP/M mit CBASIC professionell verknüpft werden: Access Manager für schnelle Dateiverwaltung, Display Manager als Maskengenerator. Diese und andere Einsatzmöglichkeiten werden beschrieben. Häufige Programmteile werden in Standardbibliotheken gespeichert und bei Bedarf neu geladen.

In Vorb. Mai 1984. Ca. 220 Seiten. **Spiralh. Ca. DM 58,–/ca. Fr. 58.–/ca. S 522,–. ISBN 3-88322-070-1**



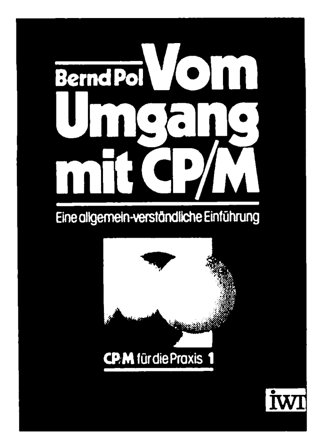
Das Buch behandelt den Einsatz von Mikrocomputern im Fremdsprachenlernprozeß. Vom Inhalt: Vokabel-Trainingsprogramm, breiter Multiple-Choice Trainingsblock, Übungen zur Erweiterung des Wortschatzes, für Alltagssituationen etc. Zahlreiche Beispiele, engl. und franz., ausführlich kommentierte Programme.

In Vorb. April 1984. Ca. 220 Seiten. **Geb. Ca. DM 58,–/ca. Fr. 58.–/ca. S 522,– ISBN 3-88322-055-8**



Das Buch ist sowohl für Programmierer als auch für Fans. Sie bekommen Einblick in das Zusammenwirken von zwei 6502-Prozessoren und den Datenverkehr über PIAs mit dem Rechner, dieses System arbeitet selbständig parallel. Ein Leckerbissen, wenn man den Computer noch besser nutzen will.

1982. 120 Seiten. Mit zahlr. Programmen. **RingO. DM 104,–/Fr. 104.–/S 936,– ISBN 3-88322-015-9**



Das Buch ist in drei Teile gegliedert: Teil 1 führt in die Eigenschaften von Computern und CP/M im besonderen ein. Aufbauend auf diesen Kenntnissen werden im 2. Teil die zentralen CP/M-Hilfsprogramme vorgestellt. Der 3. Teil geht – nach einer Einführung in die Funktionsweise des 8080-Prozessors – auf die CP/M-Systembesonderheiten ein.

1982. 386 S. Mit zahlr. prakt. Beispielen.
Geb. DM 48,–/Fr. 48.–/S. 432,–
ISBN 3-88322-004-3



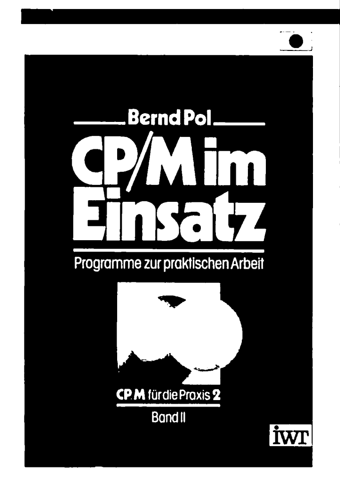
Dieser Band beschreibt wichtige Details des BDOS-Kerns und der CBIOS-Schnittstelle sowie Hinweise zur Fehlerverhütung. Weiter: Aufbau eines CBIOS-Systems, Fehlerbehandlung – Erweiterungsmöglichkeiten – Kompatibilitätsfragen zu MP/M und CP/Mplus.

In Vorb. 1984. Ca. 300 Seiten. Mit zahlreichen praktischen Beispielen.
Geb. Ca. DM 56,–/ca. Fr. 56.–/ca. S 498,–
ISBN 3-88322-006-X



Dieses Buch beschreibt an Beispielen aus der Praxis das Arbeiten mit dem wohl meistbenutzten Datenbanksystem der Welt. Es werden keine Vorkenntnisse vorausgesetzt, sondern es wird Schritt für Schritt das komplette Wissen vermittelt, das man zum Arbeiten mit einer Datenbank braucht.

In Vorb. Febr. 1984. Ca. 240 Seiten.
Geb. DM 56,–/ Fr. 56.–/ S 498,–
ISBN 3-88322-038-8



Inhalt: Macrobibliothek zur strukturierten ASM-Progr., Standard-Prozeduren, System-Schnittst. zum Abfangen aller BDOS-Fehler unter CP/M 2.2, ähnlich CP/M+, MP/M II. Standardroutinen zur zeichenorientierten I/O. Programme in strukturierter Zwischensprache setzen relozierenden Makro-ASM RMAC von D.R. voraus.

In Vorb. 1984. Ca. 360 S. Mit zahlr. ausgearb. Progr. **Geb. Ca. DM 56,–/ca. Fr. 56.–/ S 498.–. ISBN 3-88322-032-9**



Zentrale Programme für CP/M, Programme zur Nutzung der BDOS-Eigenschaften, Zentralen Datenzugriff, CBIOS-Schnittstelle, Dienstprogramme zur Erweiterung und direkten Diskettenzugriff, Routinen für Dateien. Programme kommentiert in strukturierter Zwischensprache. Relozierender Makro-ASM RMAC von DR nötig.

In Vorb. 1984. 350 Seiten. **Geb. Ca. DM 56,–/ca. Fr. 56.–/ca. S 498,–.**
ISBN 3-88322-033-7



Das Buch, auch als Einstieg in das Programmieren geeignet, zeigt dBase II als mächtige Programmiersprache für die Datenbankanwendung. Inhalt: Was ist eine Datenbank? Warum dBase II? Wie wird programmiert? Einführung: Autokostenverwaltung; für Fortgeschrittene: Literaturverwaltung, kommentierte Listings, Tips und Tricks.

In Vorb. April 1984. Ca. 240 Seiten. **Geb. Ca. DM 56,–/ca. Fr. 56,–/ca. S 498,–**
ISBN 3-88322-039-6



Dieses Buch führt Sie zur sicheren Handhabung der Tabellenkalkulation. Zahlreiche Anwendungen mit ausführlicher Beschreibung schließen sich an. Unter anderem: Werbeplanung, Reisekosten, Kapazitätsauslastung, Baufinanzierung, Reaktionszeiten.– Auch für die englische Version nutzbar.

In Vorb. Mai 1984. Ca. 250 Seiten. **Spiralh. Ca. DM 56,–/ca. Fr. 56.–/ca. S 498,–**
ISBN 3-88322-074-4



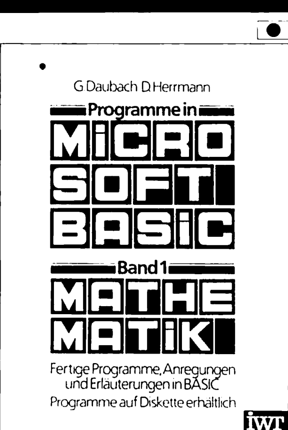
Dieses Buch wendet sich an SuperCalc-Anwender: Anfänger führt es schnell zur sicheren Handhabung, Fortgeschrittene finden ausgereifte Anwendungen. Beispiele mit ausführlicher Beschreibung machen das Buch auch für VisiCalc-Anwender interessant. Die Unterschiede zwischen VisiCalc und SuperCalc sind in einem Kapitel dargestellt.

1984. 248 Seiten. **Spiralh. DM 56,–/ Fr. 56.–/S 498,–**
ISBN 3-88322-040-X



Die Verbindung von Kalkulation, Grafik und Datenbank verhalf Lotus zu schnellem Erfolg. In diesem Buch erlernen Sie zunächst die sichere Nutzung von Kalkulation und Grafik. Zahlreiche Beispiele zeigen die Leistungsfähigkeit von 1-2-3: Umsatzprognose, Investitionsrechnung, Tilgungsplan, Private Ausgaben u.a.

In Vorb. April 1984. Ca. 250 Seiten. **Geb. Ca. DM 58,–/ca. Fr. 58.–/ca. S 522,–**
ISBN 3-88322-085-X



Dieses Buch enthält 40 mathematische Programme aus den Bereichen: Mehrregister-Arithmetik – Zahlentheorie – Kombinatorik – Algebra – Geometrie – numerische Mathematik. Neu ist die Langzahl-Arithmetik. Sie gestattet die Grundrechenarten für Zahlen bis 255 Stellen.

In Vorb. März 1984. 250 Seiten. **Spiralh. Ca. DM 48,–/ca. Fr. 48.–/ca. S 432,–**
ISBN 3-88322-077-9



Eine Hilfestellung für wirtschaftliche Entscheidungen sind Programmsammlungen, die die guten Grafik- und Farbmöglichkeiten des Computers nutzen. Diagramme, Sprites, optische Darstellungen von Simulationen werden eingesetzt, die die Ergebnisse verdeutlichen. Die finanzmathematischen Grundlagen sind zu jedem Programm beschrieben.

In Vorb. März 1984. Ca. 200 Seiten. **Spiralh. Ca. DM 48,–/ca. Fr. 48.–/ca. S 432,–**
ISBN 3-88322-076-0



Programme reichen von stöchiometrischen Berechnungen bis zur Elementedatei. Besonders genutzt werden die grafischen Möglichkeiten. Die Anwendungsbeispiele gehen von Strukturformeln, grafischen Darstellungen von Versuchsanordnungen und Funktionen bis zur 3-D Grafik im Detail.

In Vorb. Mai 1984. Ca. 220 Seiten. **Spiralh. Ca. DM 56,–/ca. Fr. 56.–/ca. S 498,–**
ISBN 3-88322-078-7

J. Hegner

Grafik in Maschinensprache auf dem Commodore 64

Das Buch gibt wertvolle Informationen über alle programmierbaren grafischen Zustände des Video-Chips 6567, wie zum Beispiel Multi Color Bit Map Mode, Smooth Scrolling oder Multi Color Character Mode. BASIC-Routinen, die grafische Muster oder mathematische Funktionen berechnen, werden von Maschinenprogrammen, wie sie zum Punkt- oder Linienzeichnen benötigt werden, unterstützt.
Die Maschinenprogramme beschleunigen die Grafikerstellung um ein Vielfaches. Alle Programme, sowohl BASIC als auch Maschinenprogramme, werden ausführlich dargestellt und zeilenweise erläutert. Teilweise werden BASIC-Programme mit Maschinenprogrammen verglichen, die die Geschwindigkeitsunterschiede ganz deutlich machen.

ISBN 3-88322-051-5